

Broken Access Control – PortSwigger Lab(s) Report

Submitted By:

Name: Arsalan Khan

Position/Role: Internee

Date: August 7, 2025

Platform:

PortSwigger Web Security Academy

Objective:

Understand and exploit a Broken Access Control vulnerability (Unprotected Admin Functionality) to gain unauthorized access to administrative functions and perform privileged actions. Practice identifying exposed administrative endpoints and evaluating their security implications.

Tools Used:

- Burp Suite
- PortSwigger Lab Environment

Broken Access Control:

Broken Access Control occurs when an application fails to properly enforce restrictions on what authenticated or unauthenticated users can do. This can allow attackers to gain unauthorized access to sensitive functions or data. In this lab, the admin panel was accessible without authentication, enabling direct execution of privileged actions such as deleting user accounts.

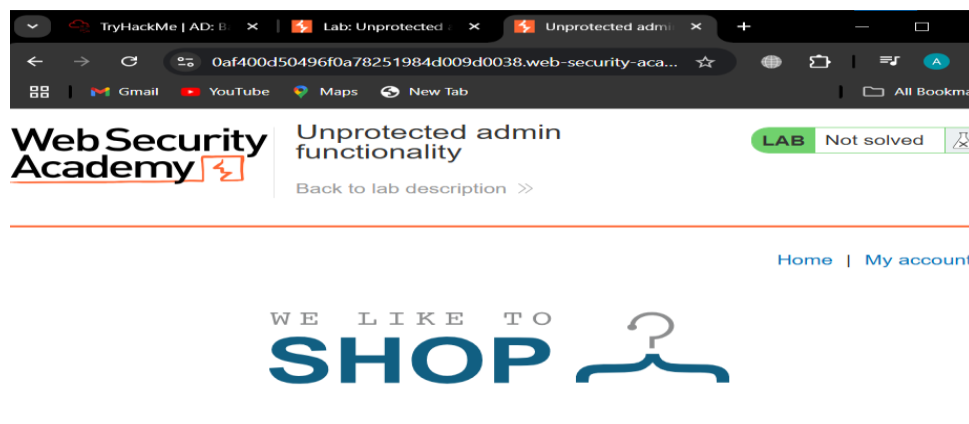
Lab 1: Unprotected Admin Functionality – Apprentice

Description:

In this lab, I identified an unprotected admin panel that was accessible without authentication. By discovering and accessing the panel, I was able to delete the user carlos.

Steps Performed:

1. Accessed the lab from the PortSwigger Web Security Academy.

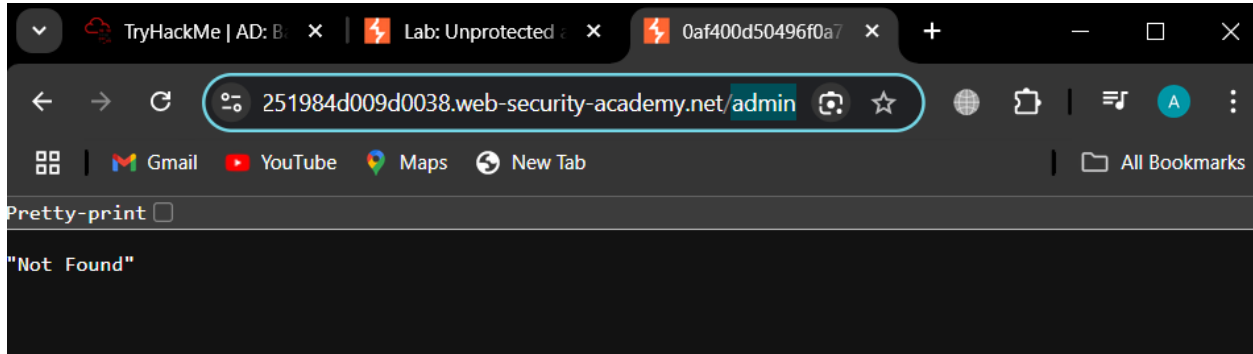


2. Navigated to the Login page to inspect possible entry points.

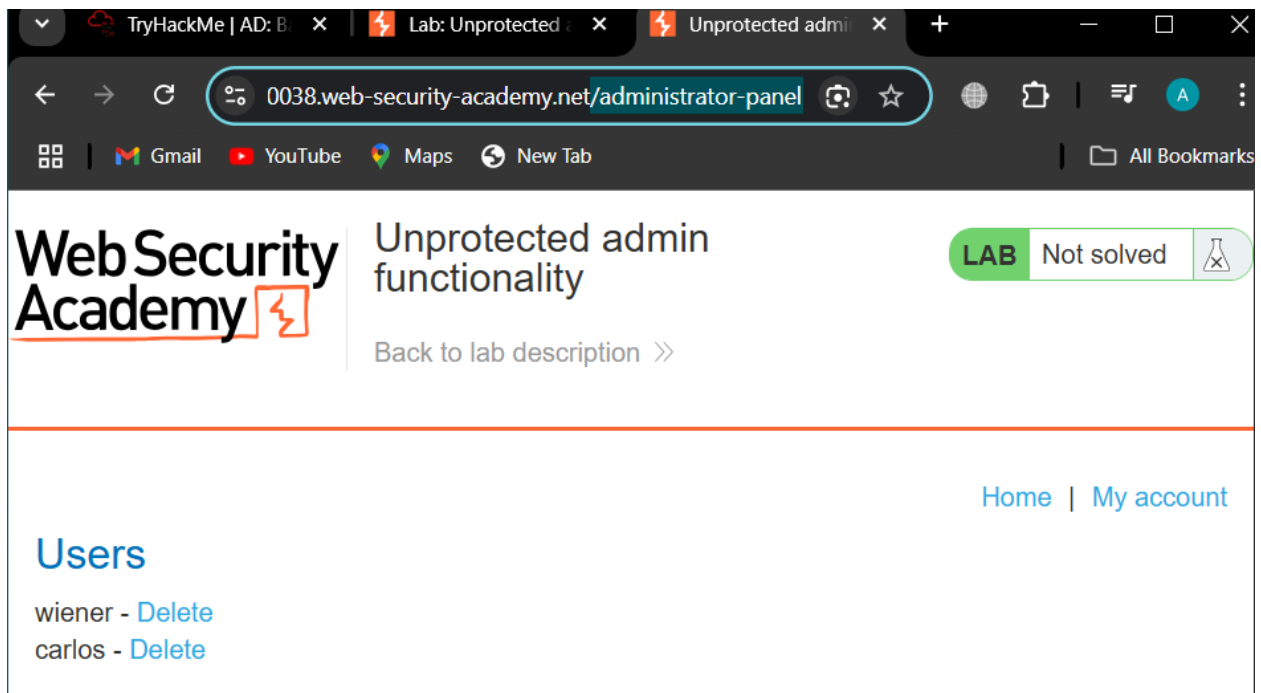
Login

A screenshot of a login form. It has a light gray background. At the top, there is a 'Username' label above a white input field with a blue border. Below that is a 'Password' label above another white input field with a blue border. At the bottom left, there is a green 'Log in' button with white text. At the top right, there are links for 'Home' and 'My account'.

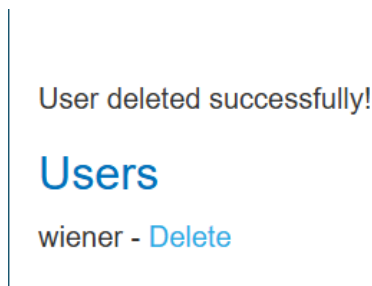
3. Tried changing the URL to /admin and similar variations, but these returned errors.



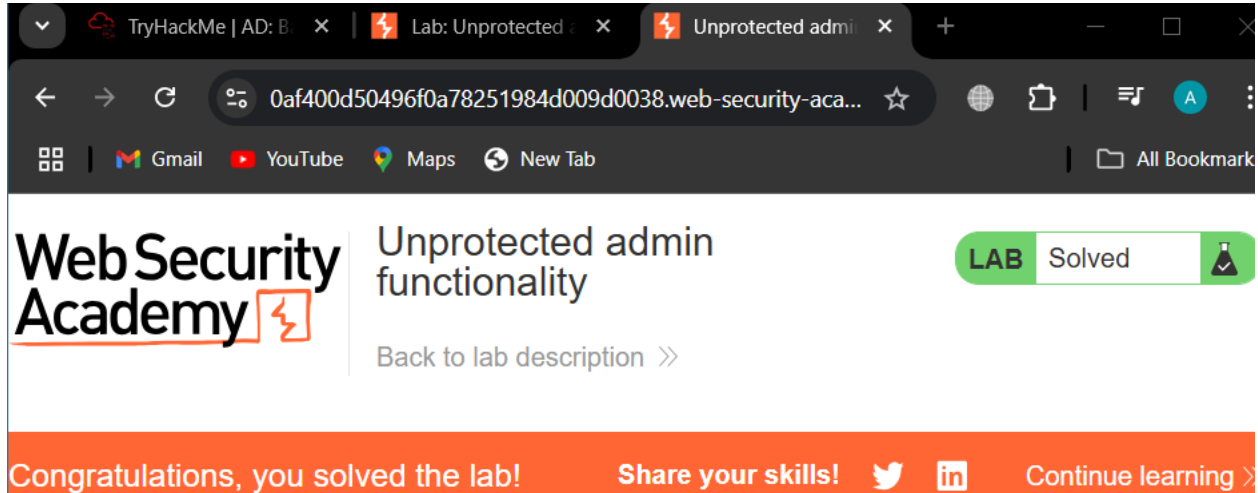
4. Appended /administrator-panel to the base URL, which successfully loaded the admin panel.



5. Found the user **carlos** in the list and clicked the delete option.)



Lab Solved:



Vulnerability:

Broken Access Control — Unprotected Admin Functionality

The application exposed an administrative panel without requiring authentication. This allowed any unauthenticated user to access privileged functions and manage user accounts.

Mitigation:

- Restrict admin panel access to authenticated users with proper role-based access control.
- Avoid using easily guessable admin paths such as `/administrator-panel`.
- Implement server-side checks to validate a user's role before granting access to administrative features.

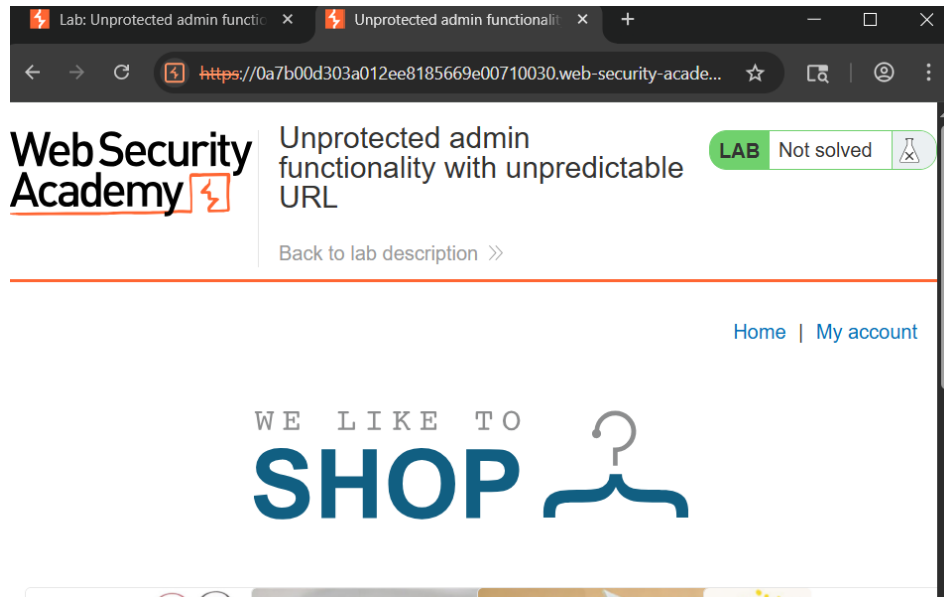
Lab 2: Unprotected Admin Functionality with Unpredictable URL

Description:

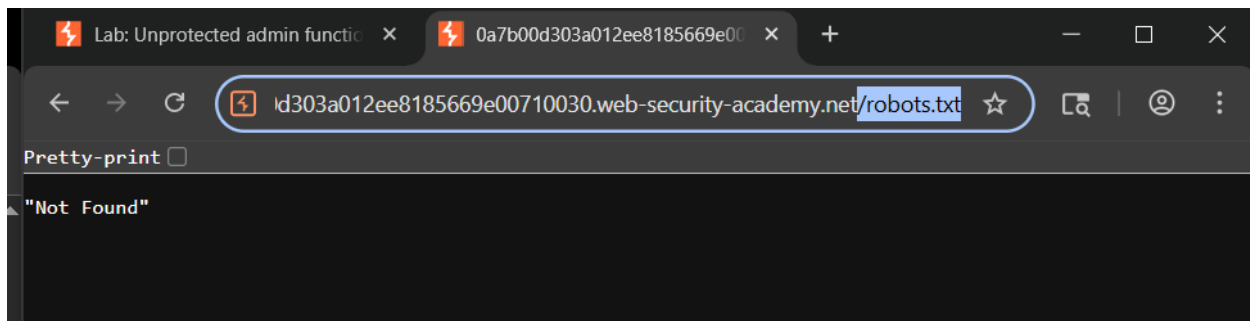
In this lab, I discovered an unprotected admin panel hidden behind an unpredictable URL. By inspecting the homepage source code, I found the path to the admin panel. Accessing this URL gave me unauthorized access to the admin interface, allowing me to delete the user **carlos**.

Steps Performed:

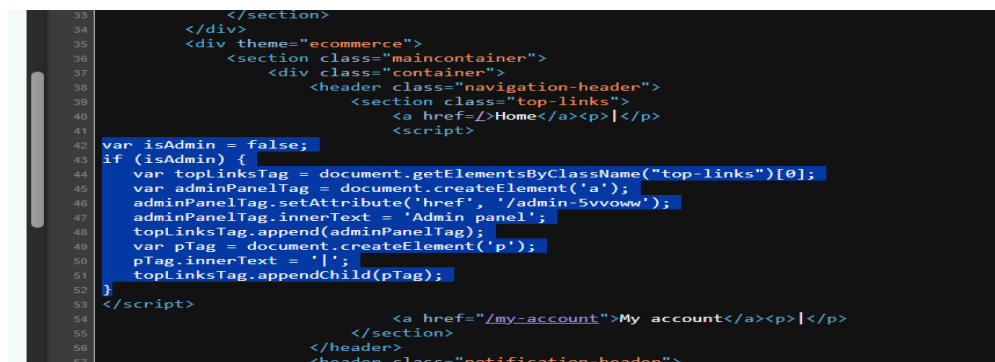
1. Accessed the lab URL.



2. Checked for the existence of a robots.txt file by appending /robots.txt to the URL, but the file was not present.



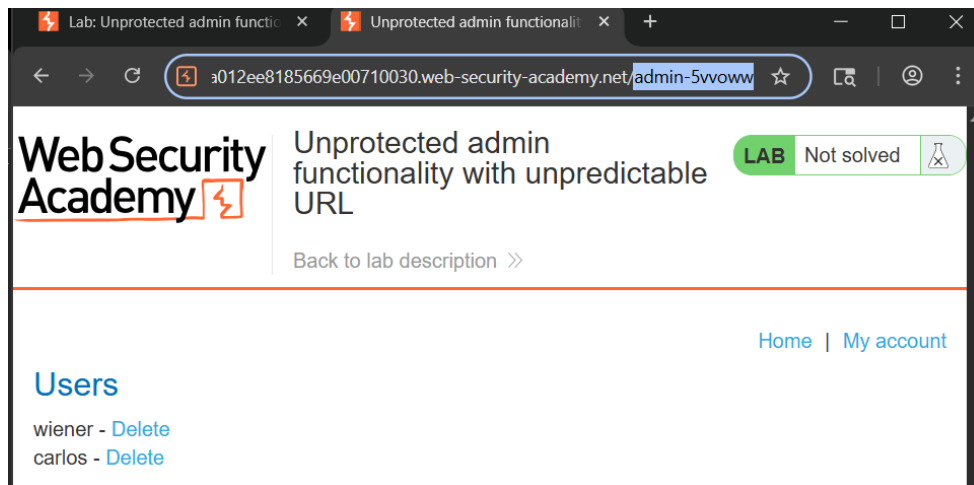
1. Loaded the homepage and viewed its source code in the browser. Found JavaScript logic containing the admin directory path.



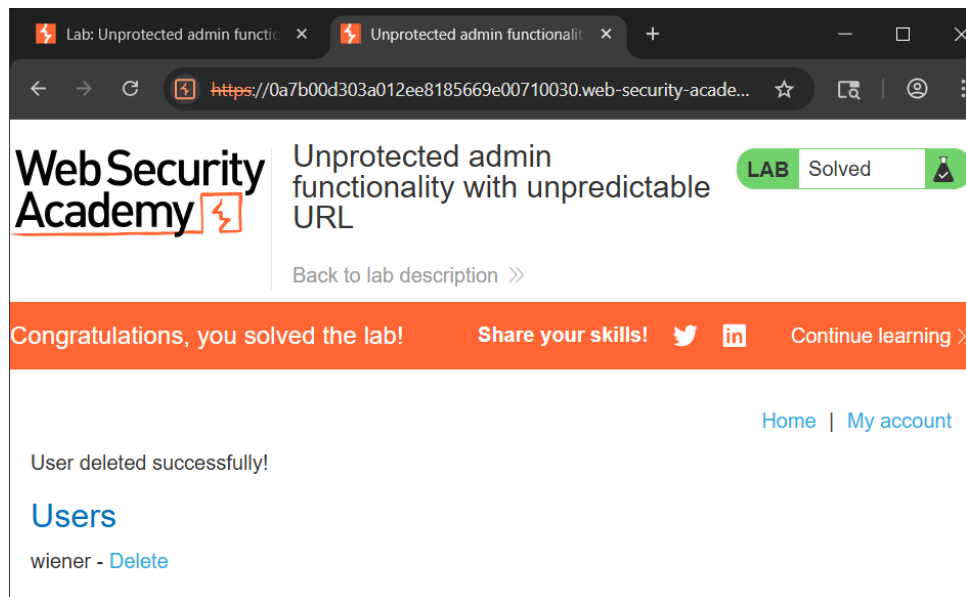
2. Copied the admin directory path from the source and appended it to the base URL.

```
var isAdmin = false;
if (isAdmin) {
  var topLinksTag = document.getElementsByClassName("top-links")[0];
  var adminPanelTag = document.createElement('a');
  adminPanelTag.setAttribute('href', '/admin-5vvoww');
  adminPanelTag.innerText = 'Admin panel';
  topLinksTag.append(adminPanelTag);
  var pTag = document.createElement('p');
```

3. Successfully accessed the admin panel using the discovered URL.



4. Deleted the user carlos via the admin panel.



Vulnerability:

Broken Access Control – Unprotected Admin Functionality with Information Disclosure

The application exposes an administrative panel at a hidden URL which is not protected by authentication. The URL was discoverable through source code inspection, allowing unauthorized users to perform privileged actions such as deleting user accounts.

Mitigation:

- Enforce authentication and authorization on all administrative pages.
- Avoid exposing sensitive URLs in client-side code.
- Use secure methods to hide administrative endpoints and ensure proper access controls.
- Regularly review client-side code for accidental information disclosure.

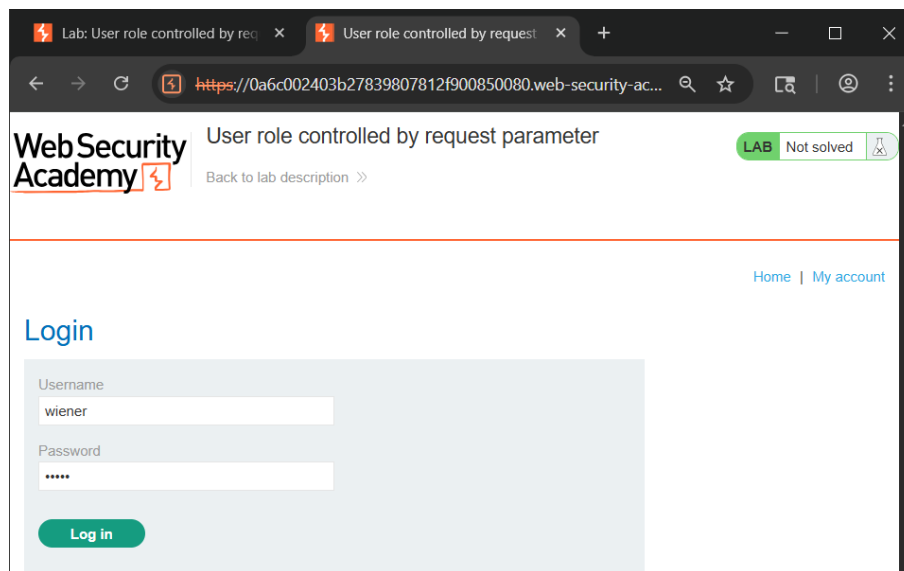
Lab 3: User Role Controlled by Request Parameter –

Description:

In this lab, I exploited a broken access control vulnerability where user role information was controlled by a client-side cookie. By modifying the cookie value from `false` to `true`, I was able to gain unauthorized admin access and delete the user **carlos**.

Steps Performed:

1. Accessed the lab and logged in using the provided credentials.



- Intercepted the /my-account request using Burp Suite Proxy.

https://0a6c002403b27839...	GET	/resources/academyHeader/images/ps-1...		200	342	HTML	svg
https://0a6c002403b27839...	GET	/academyLabHeader	✓	400	130	text	
https://0a6c002403b27839...	GET	/my-account		302	86		
https://0a6c002403b27839...	GET	/login		200	3271	HTML	
https://0a6c002403b27839...	GET	/academyLabHeader	✓	400	130	text	
https://0a6c002403b27839...	POST	/login	✓				

- Sent the intercepted request to Burp Repeater for testing.

The screenshot shows the Burp Suite Repeater tab. At the top, there's a toolbar with 'Send', a settings gear, 'Cancel', and navigation arrows. Below the toolbar, the 'Request' tab is active, showing a 'Pretty' view of a GET request. The request details are as follows:

```
1 GET /my-account?id=wiener HTTP/2
2 Host: 0a6c002403b27839807812f900850080.web-security-academy.net
3 Cookie: Admin=false; session=ECGqD6tFpPfhKQVpjKSIeKz5hWwlcilN
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
8 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
```

The 'Response' tab is also visible, showing a 'Pretty' view of the response:

```
1 HTTP/2
2 Content-Type: text/html; charset=utf-8
3 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
4 X-Frame-Options: DENY
5 Content-Security-Policy: frame-ancestors 'none'
```

- Modified the cookie parameter admin from false to true in the Repeater request.

The screenshot shows the Burp Suite Repeater tab with the 'Request' tab active. The 'Pretty' view of the GET request is shown, with the 'Cookie' line modified to 'Admin=true; session=ECGqD6tFpPfhKQVpjKSIeKz5hWwlcilN'. The rest of the request details are the same as in the previous screenshot.

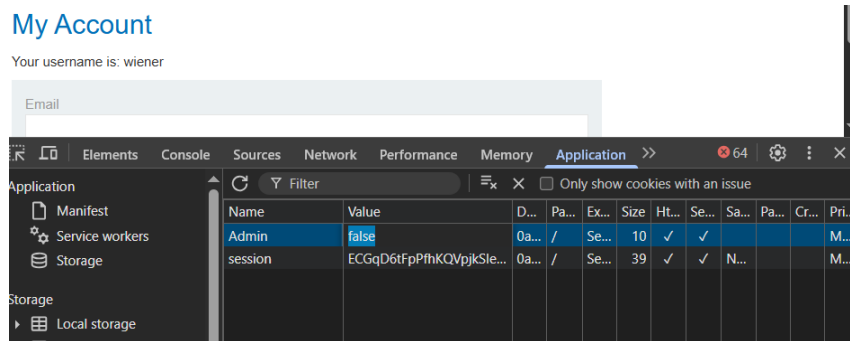
```
1 GET /my-account?id=wiener HTTP/2
2 Host: 0a6c002403b27839807812f900850080.web-security-academy.net
3 Cookie: Admin=true; session=ECGqD6tFpPfhKQVpjKSIeKz5hWwlcilN
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
8 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
```

- Sent the modified request and observed the response indicating admin access was granted.

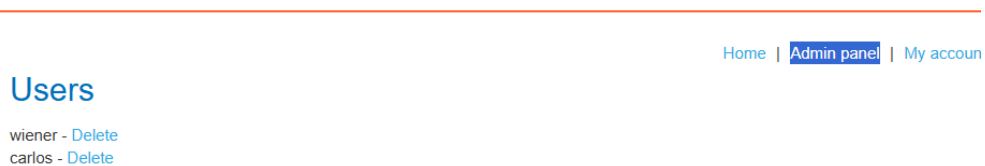
Response

	Pretty	Raw	Hex	Render
16		<code>Home</code>		
		<code></code>		
		<code><p></code>		
		<code> </code>		
		<code></p></code>		
17		<code></code>		
		<code>Admin panel</code>		
		<code></code>		
		<code><p></code>		
		<code> </code>		

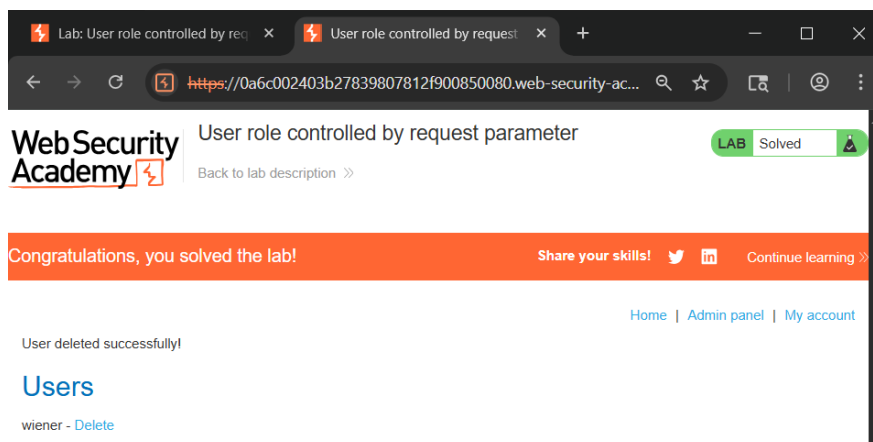
- To reflect this change in the browser, opened the browser's Developer Tools → Application → Cookies. Edited the `admin` cookie value to `true`.



- Reloaded the My Account page, which now displayed the admin interface.



- Deleted the user **carlos** using the admin interface.



Vulnerability:

Broken Access Control via Client-Side Role Manipulation

The application relies on a client-side cookie to determine user roles, which can be tampered with by attackers. Since the server does not properly verify the role on each request, attackers can escalate privileges by simply changing the cookie value and gain unauthorized access to admin features.

Mitigation:

- Store user roles securely on the server side and avoid trusting client-side data for authorization.
 - Implement proper server-side checks for every privileged request.
 - Use signed tokens or secure session management to prevent tampering.
 - Regularly audit role management implementations for vulnerabilities.
-

Lab 3: User Role Can Be Modified in User Profile –

Description:

In this lab, I exploited a vulnerability allowing me to modify my user role via the profile update feature. By intercepting and tampering with the email change request, I elevated my role ID from 1 to 2 (administrator). This granted access to the admin panel where I deleted the user **carlos**.

Steps Performed:

1. Accessed the lab and logged in using the given credentials.
2. Navigated to My Account page on the browser and changed the email address to:

My Account

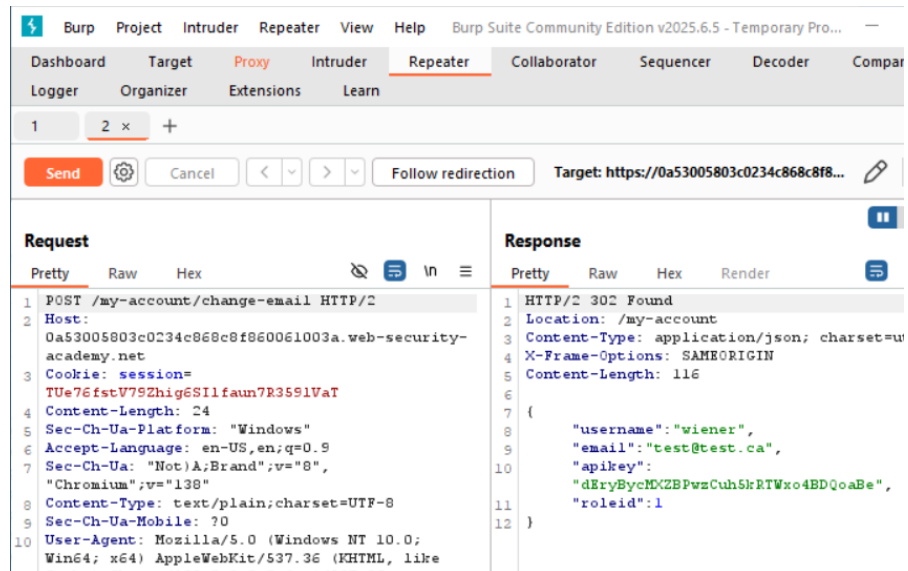
Your username is: wiener

Your email is: wiener@normal-user.net

Email

Update email

- Intercepted the my-account/change-email request in Burp Suite Proxy. Sent the intercepted request to Burp Repeater for modification.



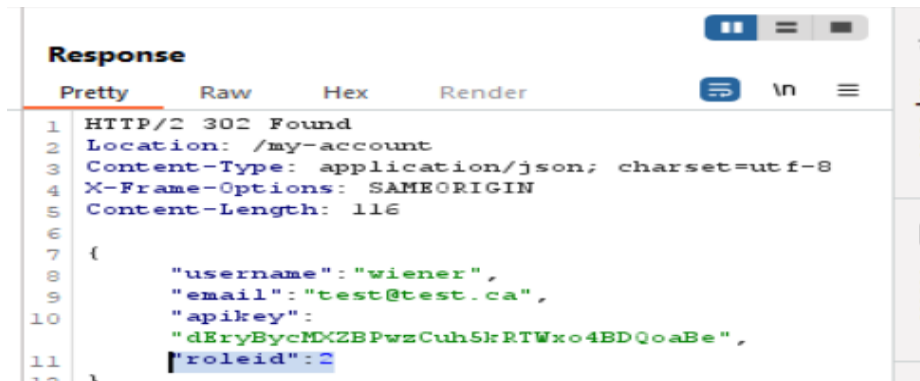
- Observed the response containing user details, including `"roleid": 1`.

```
1 HTTP/2 302 Found
2 Location: /my-account
3 Content-Type: application/json; charset=utf-8
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 116
6
7 {
8   "username": "wiener",
9   "email": "test@test.ca",
10  "apikey":
11    "dEryBycMXZBPwzCuh5kRTWxo4BDQoaBe",
12  "roleid": 1
13 }
```

- Edited the JSON request body by adding a comma and `"roleid": 2` after the email field.

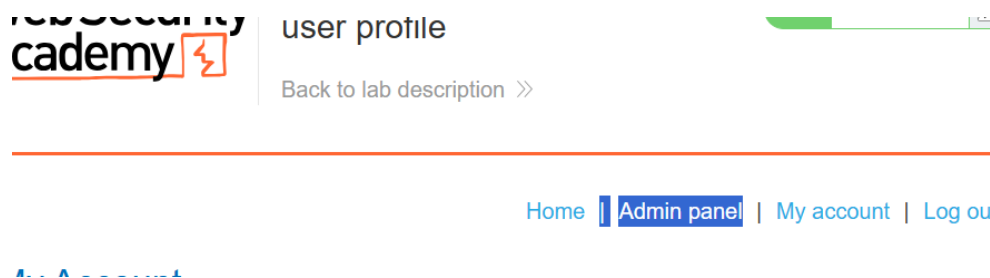
```
--
https://0a53005803c0234c868c8f860061003a.web-s
ecurity-academy.net/my-account?id=wiener
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 {
21   "email": "test@test.ca",
22   "roleid": 2
23 }
```

6. Resent the modified request and received a response showing "roleid": 2, confirming role escalation.

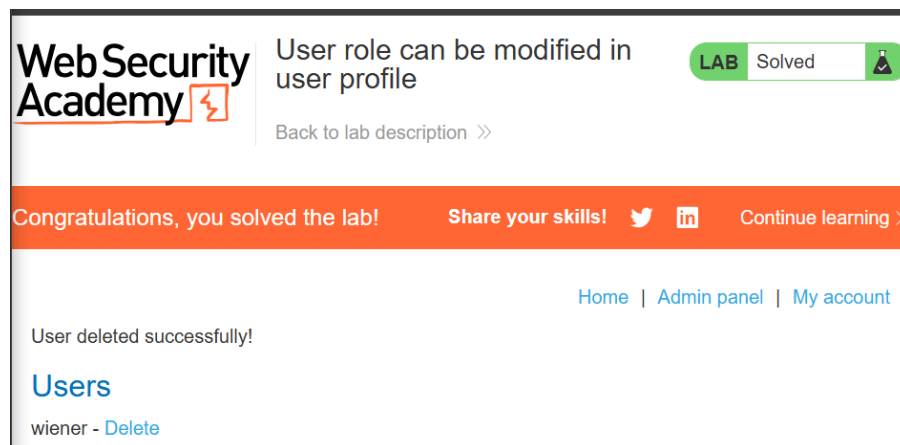


```
Response
Pretty Raw Hex Render
1 HTTP/2 302 Found
2 Location: /my-account
3 Content-Type: application/json; charset=utf-8
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 116
6
7 {
8   "username": "wiener",
9   "email": "test@test.ca",
10  "apikey": "dEryBycMDXZBPwzCuh5kRTWxo4BDQoaBe",
11  "roleid": 2
12 }
```

7. Reloaded the browser, which now displayed the admin panel link.



8. Accessed the admin panel and deleted the user carlos.



Vulnerability:

Broken Access Control via User-Modifiable Role ID in Profile Update

The application trusts user-submitted data to update critical fields like `roleid` without proper validation or authorization checks. This allows attackers to escalate privileges by changing their role and accessing restricted admin features.

Mitigation:

- Validate and enforce user roles strictly on the server side; do not accept role changes from client requests.
- Remove sensitive parameters like `roleid` from user-controllable inputs.
- Use server-managed sessions to maintain user role information.
- Monitor for suspicious privilege escalation activity in logs.

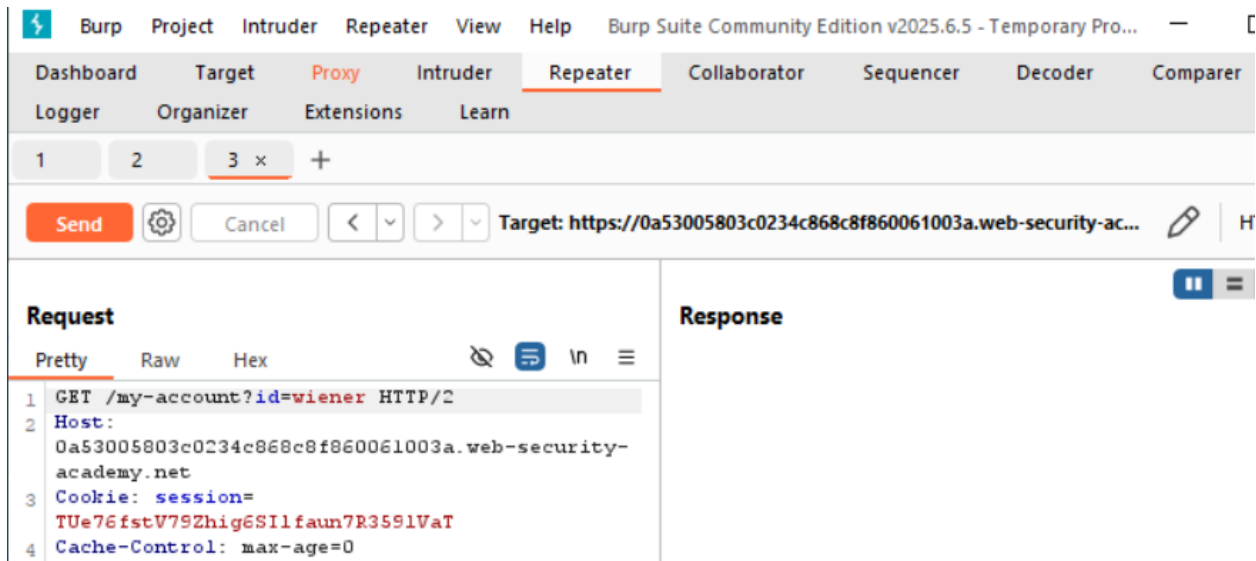
Lab 4: User ID Controlled by Request Parameter

Description:

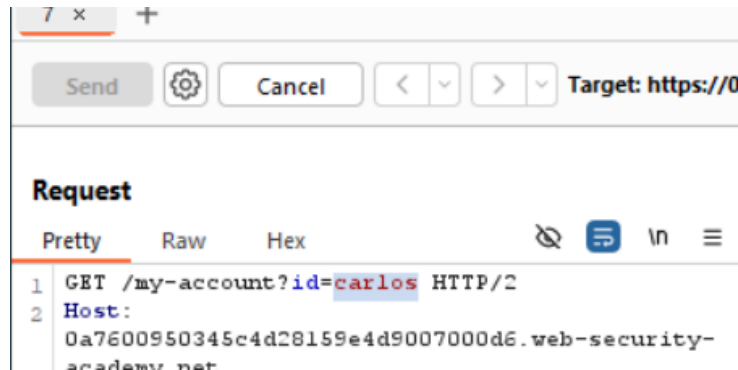
In this lab, I exploited a horizontal privilege escalation vulnerability by manipulating the user ID parameter in the account page request. By changing the user ID from my own to another user's (`carlos`), I was able to retrieve Carlos's API key and solve the lab.

Steps Performed:

1. Accessed the lab and logged in using the provided credentials (`wiener:peter`). Navigated to the My Account tab in the application.
2. Intercepted the account page request using Burp Suite Proxy. Sent the intercepted request to Burp Repeater for testing



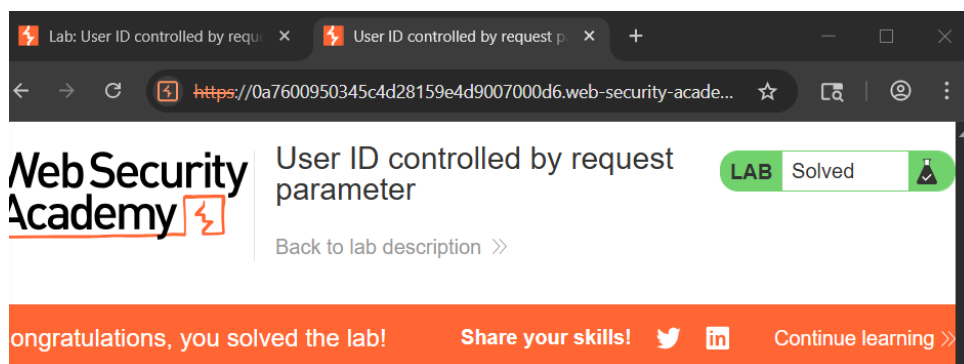
3. Modified the user ID parameter in the request from wiener to carlos.



4. Sent the modified request and received a response containing Carlos's username and API key.

```
My Account
</h1>
<div id=account-content>
  <p>
    Your username is: carlos
  </p>
  <div>
    Your API Key is: 70J0wTynIk9zyJ7fR5RHn7e4HvXcBFcZ
  </div>
<br/>
<form class="login-form" name="change-email-form" action="
/my-account/change-email" method="POST">
  <label>
    Email
```

5. Copied Carlos's API key and submitted it as the lab solution.
6. Lab marked as solved.



Vulnerability:

Horizontal Privilege Escalation via User ID Parameter Manipulation

The application fails to verify that the authenticated user is authorized to access the requested account data. This allows attackers to change user identifiers in requests to access other users' sensitive information.

Mitigation:

- Implement strict server-side authorization checks to ensure users can only access their own data.
 - Validate all user-controllable parameters against the authenticated user's identity.
 - Use session-based identifiers instead of user-submitted parameters for sensitive operations.
 - Monitor access logs for abnormal access patterns.
-

Summary of Labs on Broken Access Control

1. **Unprotected Admin Functionality**
Discovered an unprotected admin panel URL disclosed in robots.txt. Accessing this URL allowed deleting the user *carlos* without authentication.
 2. **Unprotected Admin Functionality with Unpredictable URL**
Found the admin panel URL hidden in JavaScript on the homepage. By appending this URL, gained admin access and deleted *carlos*.
 3. **User Role Controlled by Request Parameter**
Modified a client-side cookie controlling user roles from false to true, escalating privileges to admin and deleting *carlos*.
 4. **User Role Can Be Modified in User Profile**
Intercepted and altered a profile update request to change the roleid to admin level, granting access to the admin panel and deleting *carlos*.
 5. **User ID Controlled by Request Parameter**
Performed horizontal privilege escalation by changing the user ID parameter in an account request to access carlos's API key and retrieve it successfully.
-