1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

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

APPRENTICE

LAB | Not solved

This lab contains a SQL injection vulnerability in the product category filter. When the user selects a category, the application carries out a SQL query like the following:

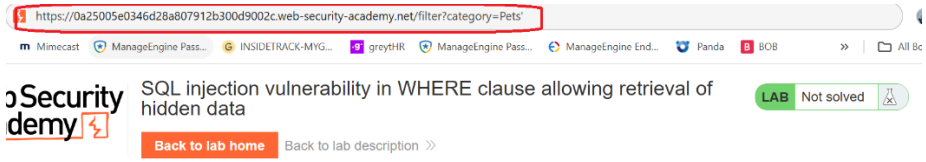
```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

To solve the lab, perform a SQL injection attack that causes the application to display one or more unreleased products.

ACCESS THE LAB

- Access the lab and select any category-URL seen as follows

- Modify the URL by adding a ' after the category value



- Got following error-means, our input is breaking the underlying SQL query.

- We now inject:

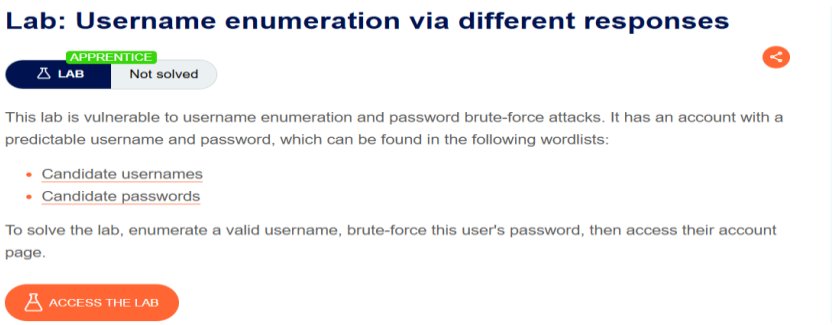
pets' OR 1=1—

- Component Meaning
  ' Closes the original SQL string
  OR 1=1 Always TRUE — returns all rows
  -- Comments out the rest of the SQL query

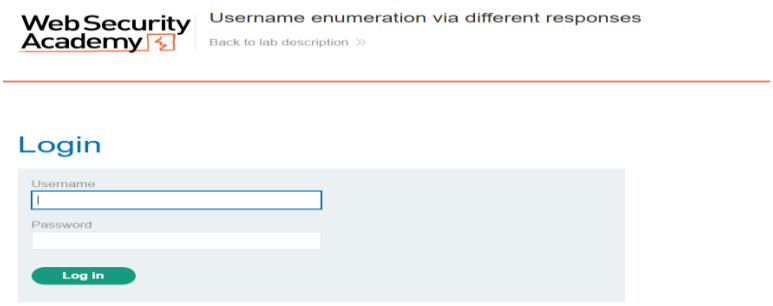
  The SQL becomes:

  SELECT * FROM products
  WHERE category = 'Gifts' OR 1=1--' AND released = 1
  Everything after -- is ignored.
  1=1 forces all products, including hidden ones, to be returned.
- After the injection, the product list should suddenly include:
  Items from other categories, and
  Items previously not visible (hidden/unreleased items)

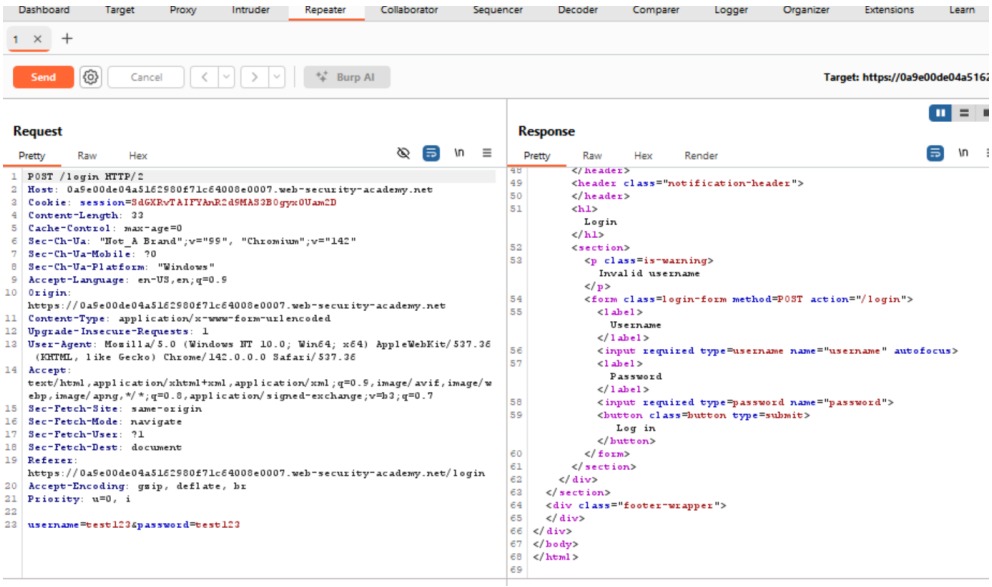
2. Username enumeration via different responses



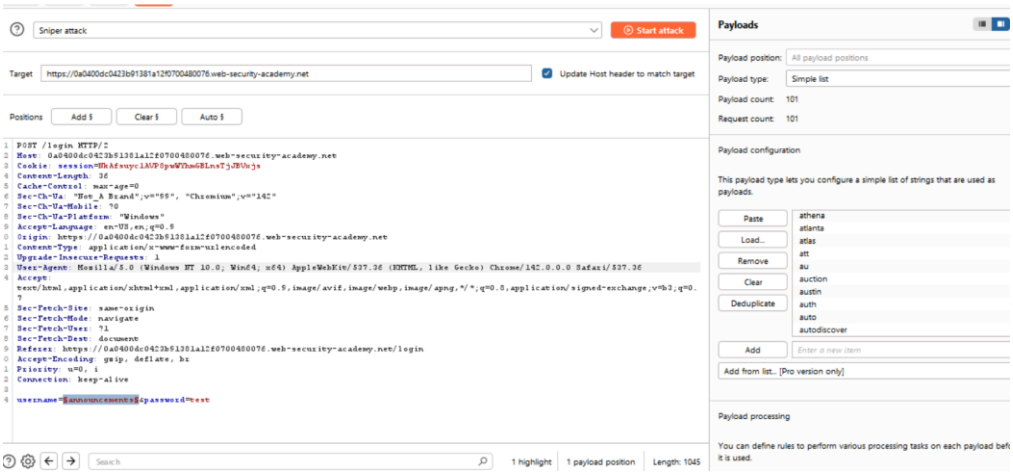
- Start the lab and open the login page:
  /login



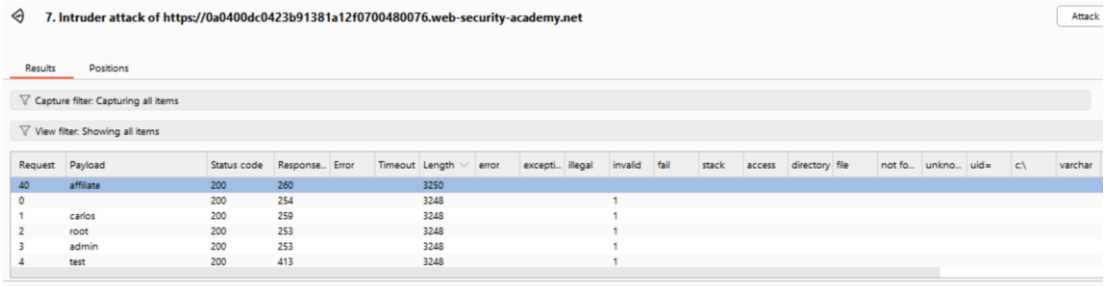
- Open BurpSuite →Turn Intercept ON→In the browser, enter any random values→ click login
- BurpSuite captured a POST request with given username and password→ Forward it or send it to Repeater→go to repeater→ see invalid username response



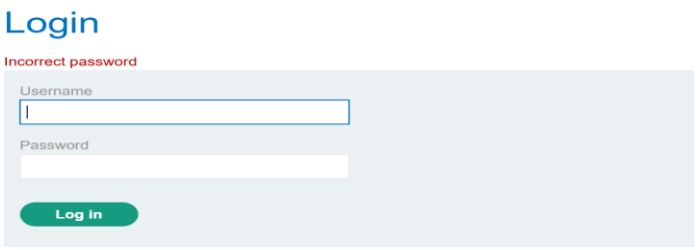
- In Repeater → Right-click → Send to Intruder
  Highlight only the username= value:
  username= §administrator§&password=test123
  Make sure that Sniper attack is selected→ Simple list payload type is selected → paste the list of candidate usernames. Finally, click Start attack. The attack will start in a new window.



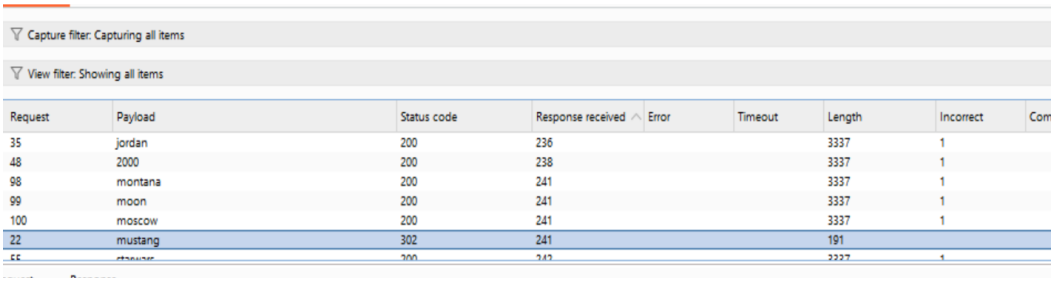
one of the entries is longer than the others.

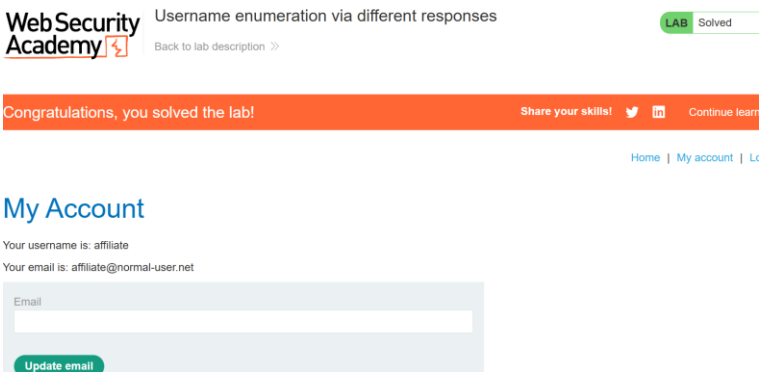


Use the one with longer length and any password. Got a different result

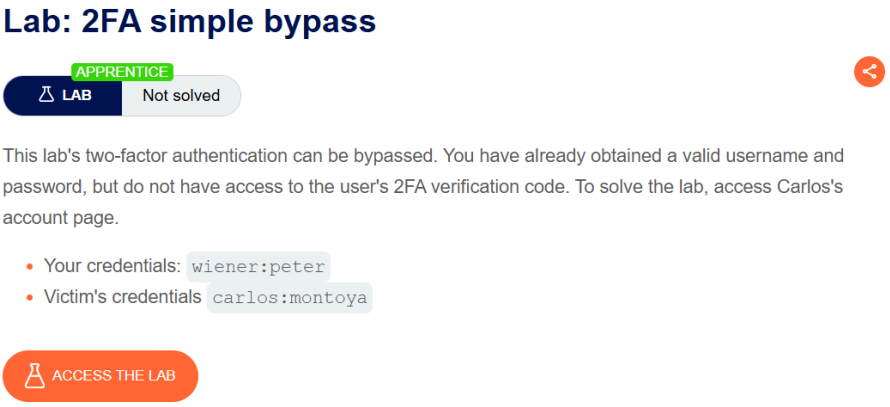


add the founded username and do the same for password select and use the one with status code 302

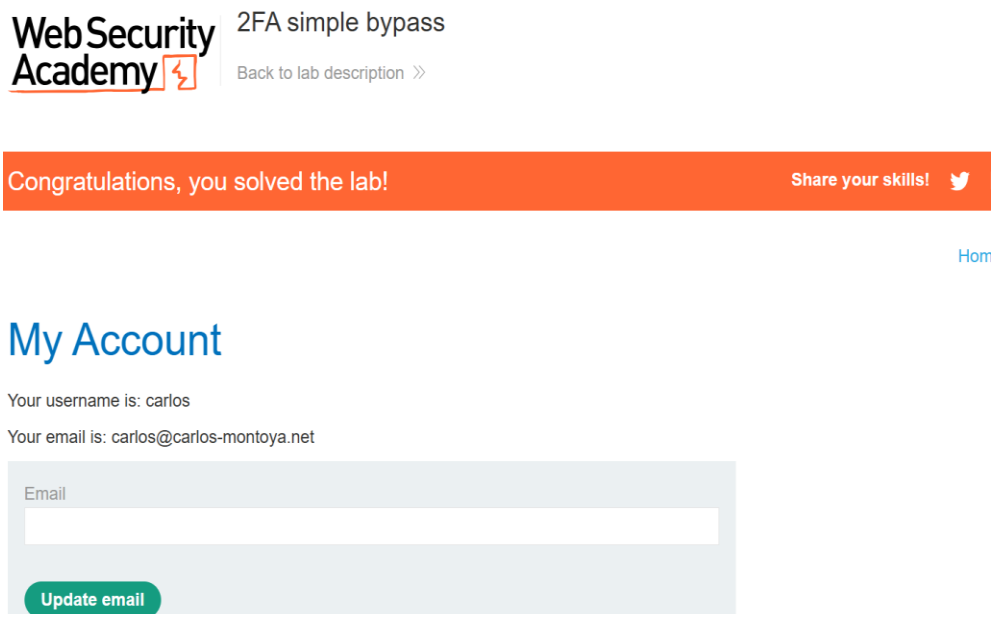




3. *2FA simple bypass*



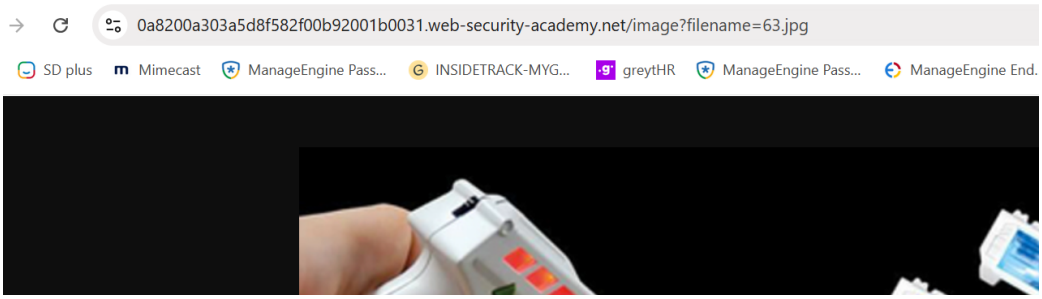
- Log in to your own account. Your 2FA verification code will be sent to you by email. Click the Email client button to access your emails.
- Go to your account page and make a note of the URL.
- Log out of your account.
- Log in using the victim's credentials.
- When prompted for the verification code, manually change the URL to navigate to /my-account. The lab is solved when the page loads.

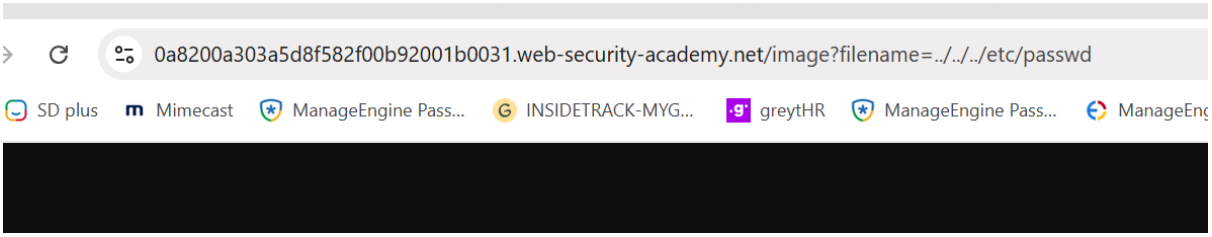


4. File path traversal, simple case

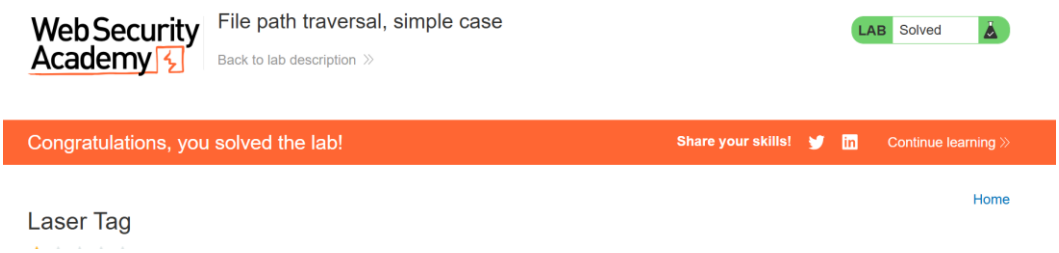
Modify the filename parameter, giving it the value:

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

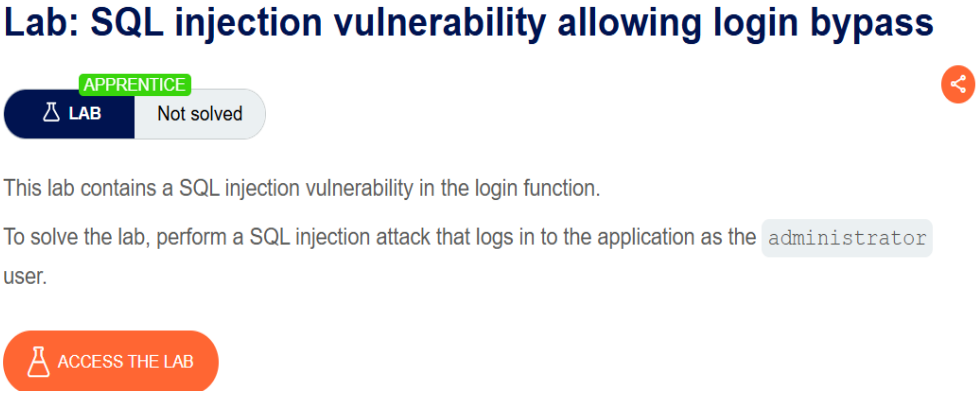




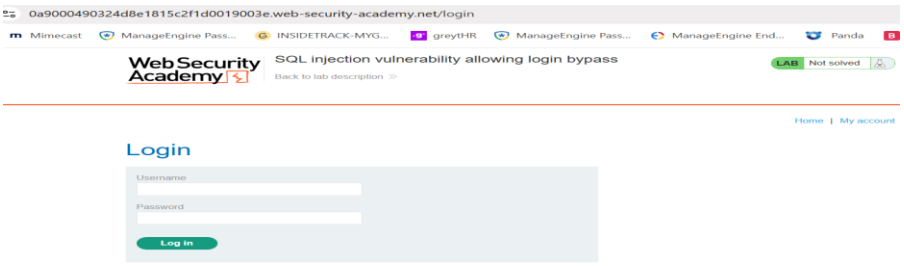
Observe that the response contains the contents of the /etc/passwd file.



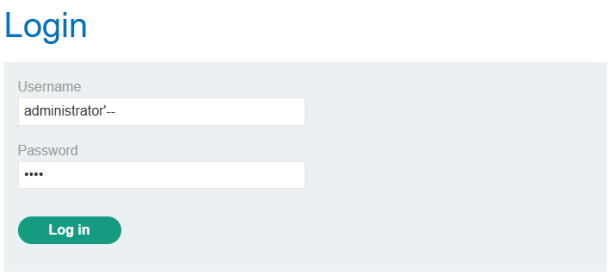
5. SQL injection vulnerability allowing login bypass



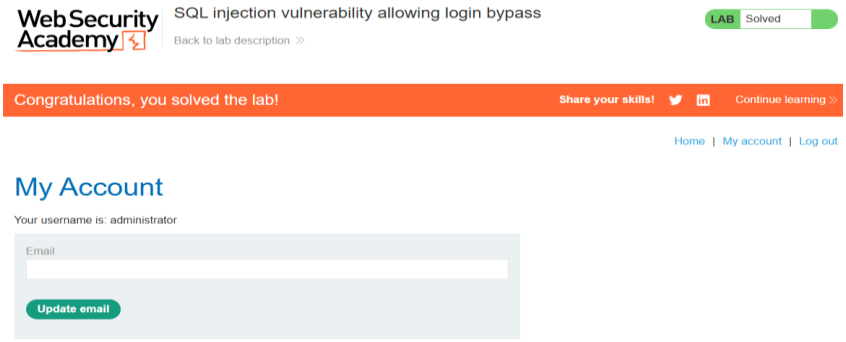
- Access the lab→ My Account



- Go to the username field
  Enter this payload:
  administrator'—
- Go to the password field
  Enter anything (doesn't matter):
  Test



- Click login

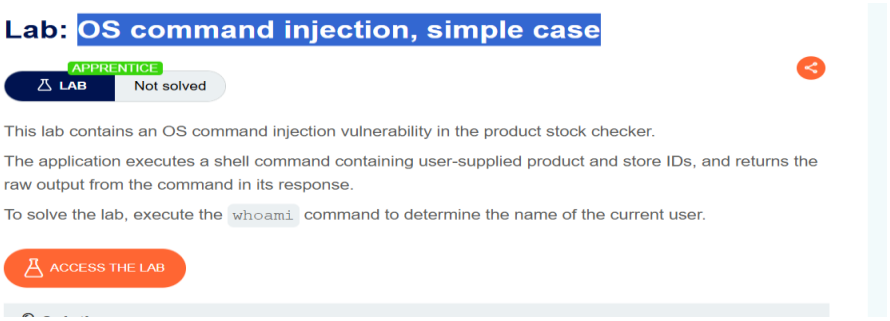


- Our input makes the backend query look like:
  *SELECT * FROM users WHERE username='administrator'--' AND password='test';*

Everything after -- is treated as a comment, so the password check is ignored.

- Effectively becomes:
  *SELECT * FROM users WHERE username='administrator';*
- So we got logged in without the password.

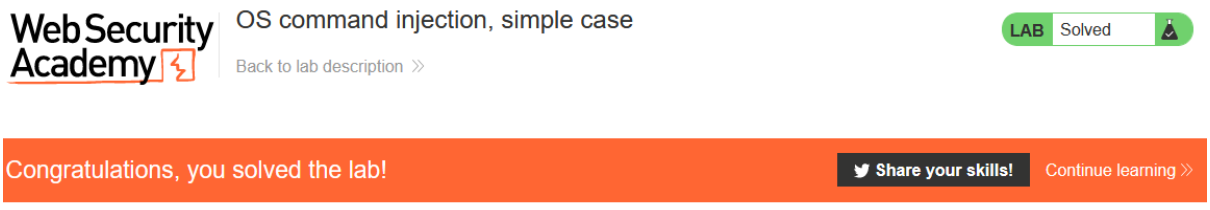
6. *OS command injection, simple case*

- 

- Intercepted request sends to repeater

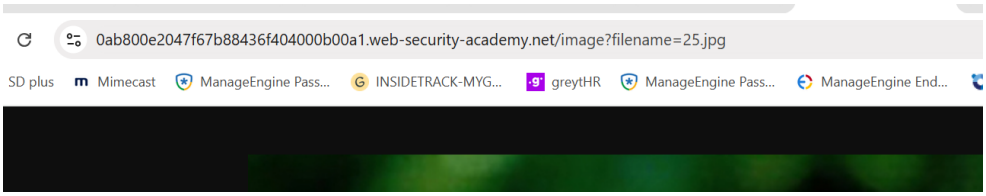
- Modified the storeID with 1|whoami

- Solved the lab

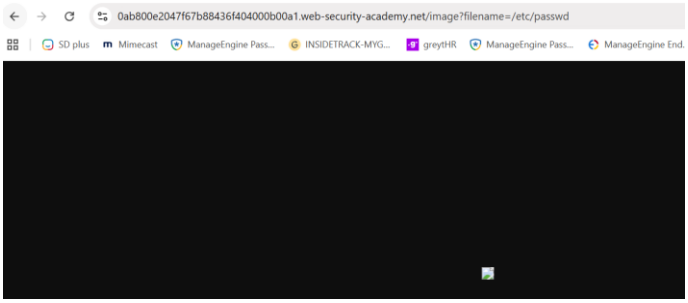


7. *File path traversal, traversal sequences blocked with absolute path bypass*

- Modify the filename parameter, giving it the value /etc/passwd.

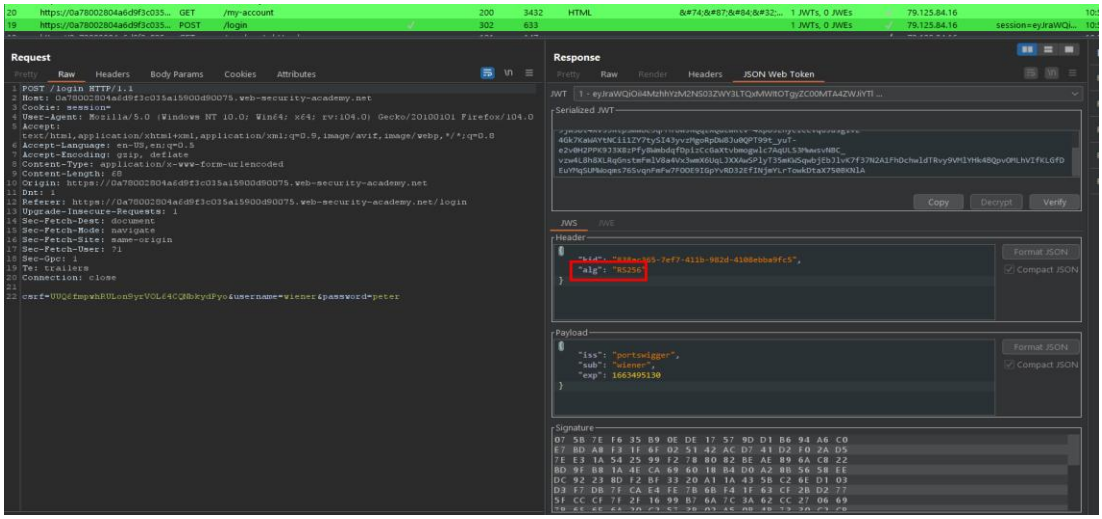


- Observe that the response contains the contents of the /etc/passwd file.



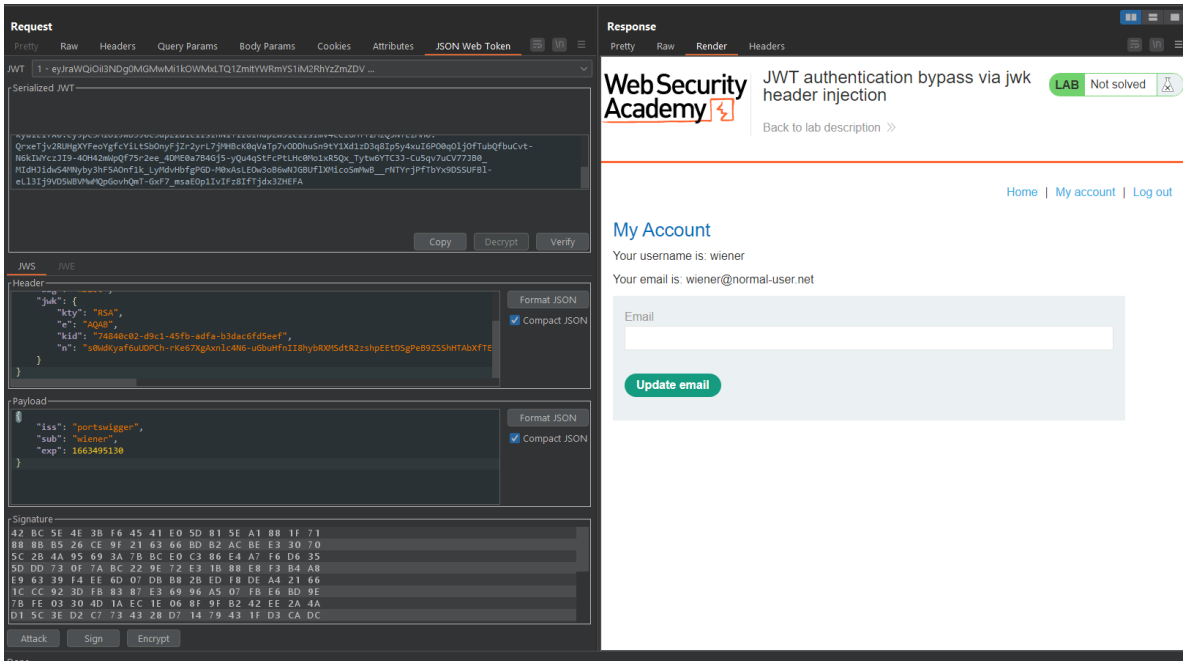
8. *JWT authentication bypass via jwk header injection*

1. In the lab, log in to your own account and send the GET /my-account?id=wiener request to Burp Repeater.
2. In Burp Repeater, change the path to /admin and send the request. Observe that the admin panel is only accessible when logged in as the administrator user

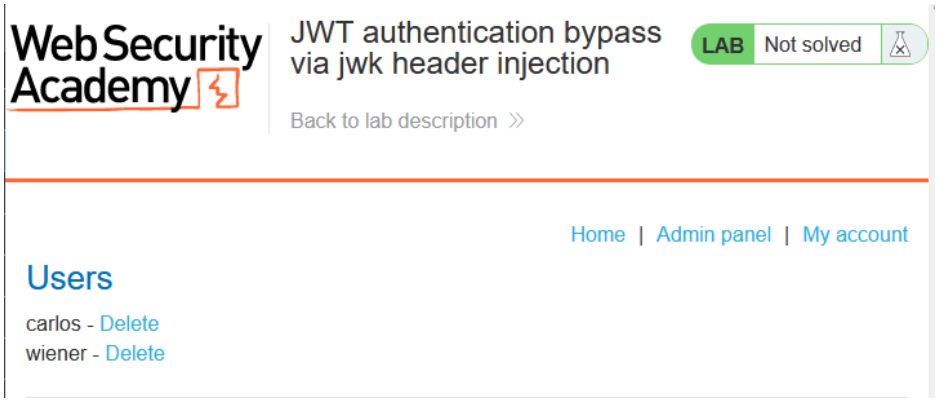


generate a new *RSA key* in the *JWT Editor*

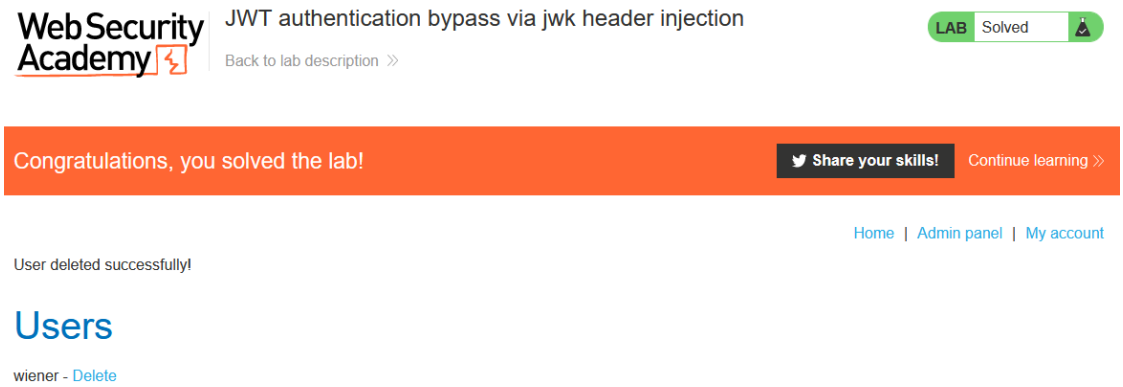
send the request to /my-account to Repeater and select the Attack -> Embedd JWK option. select the RSA signing key



refresh the /admin page.

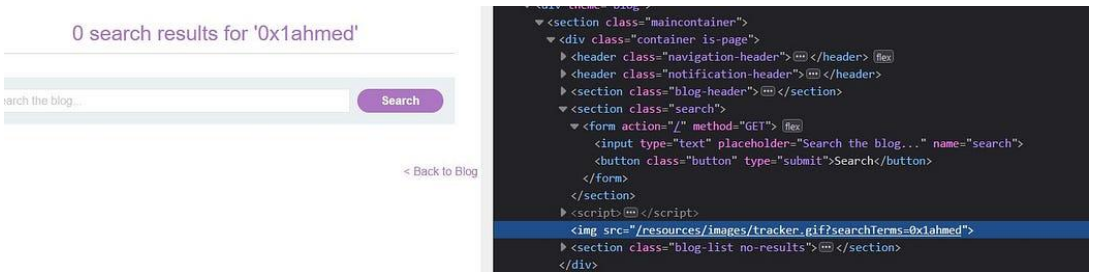


After clicking on the Delete link for user carlos



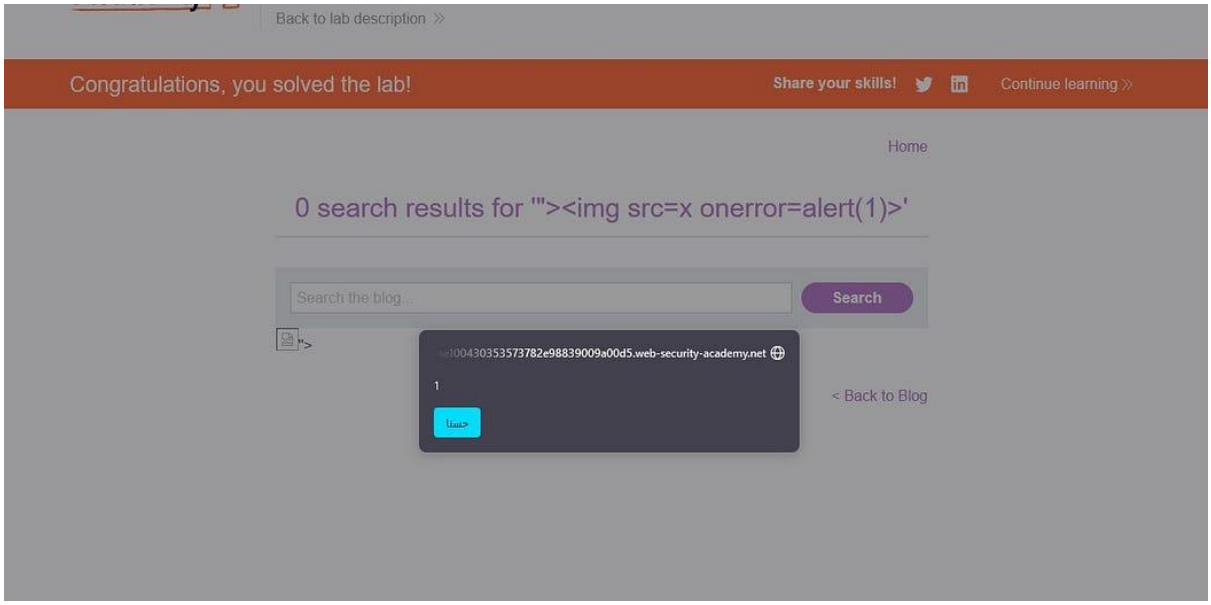
9. ***DOM XSS in document.write sink using source location.search***

1. Enter a random alphanumeric string into the search box.
2. Right-click and inspect the element, and observe that your random string has been placed inside an img src attribute.



3. Break out of the img attribute by searching for:

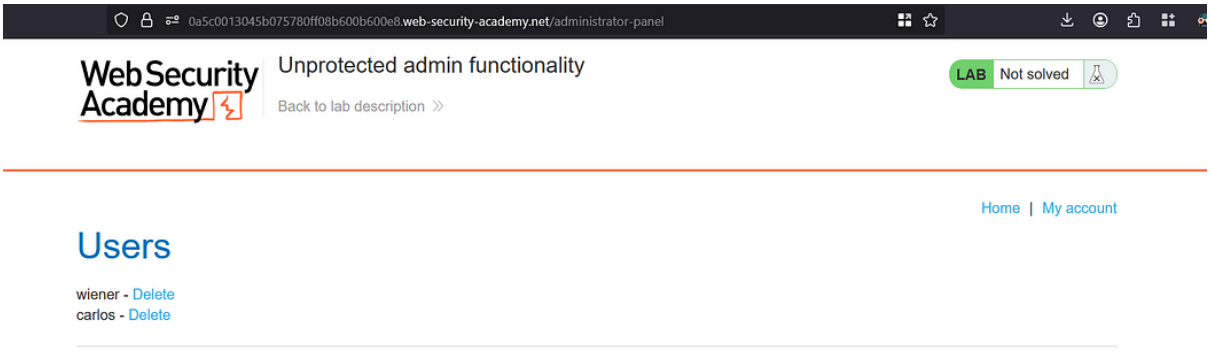
"><svg onload=alert(1)>



10. ***Unprotected admin functionality***

1. Go to the lab and view robots.txt by appending /robots.txt to the lab URL. Notice that the Disallow line discloses the path to the admin panel.

2. In the URL bar, replace /robots.txt with /administrator-panel to load the admin panel.



3. Delete carlos.