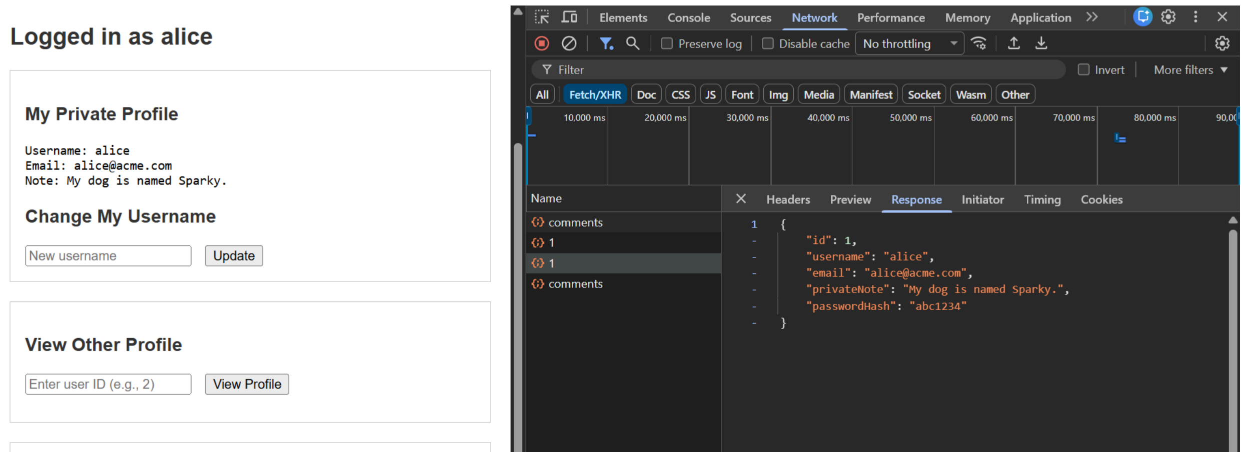
AJAX Lab

Agenda	To explore and learn about vulnerabilities associated with AJAX and the web
Lab	Given a vulnerable website, find and exploit vulnerabilities and propose remediation solutions.
Explored vulnerabilities	1. EDE (Excessive Data Exposure) 2. IDOR (Insecure Direct Object Reference) 3. XSS (Cross Site Scripting) 4. CSRF (Cross Site Request Forgery)
Codebase	https://github.com/syedabrar-afk/labs/tree/main/ajax

EDE (Excessive Data Exposure)

- EDE represents the scenario in which the server sends data more than needed. In other words, instead of only sending the required information, the server sends excessive information which is not required to be displayed but can be useful for the sight of an attacker as EDE can also expose confidential information.
- In the website - When the user logs in, or searches for another user, the server simply sends the entire user object including confidential credentials such as the hashed password, though this excessive information is not required.



```
Username: alice
Email: alice@acme.com
Note: My dog is named Sparky.
```

```
{
    "id": 1,
    "username": "alice",
    "email": "alice@acme.com",
    "privateNote": "My dog is named Sparky.",
    "passwordHash": "abc1234"
}
```

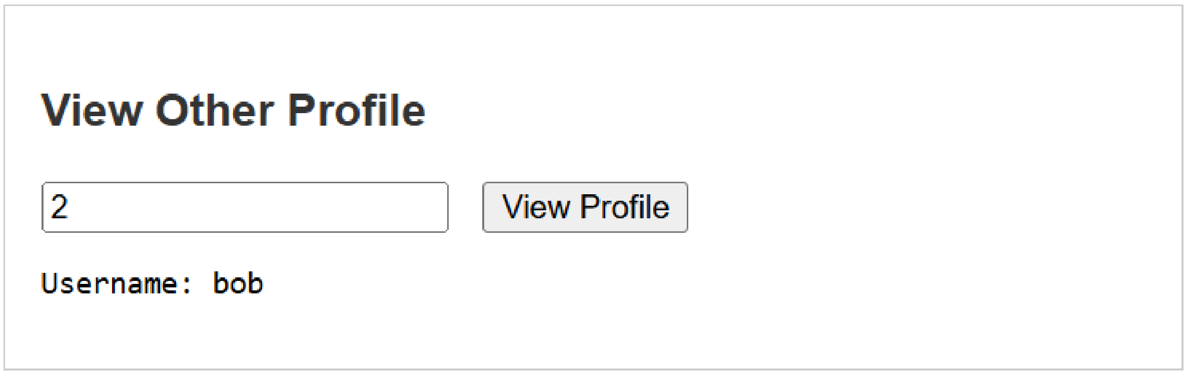
Fig.1. The DOM only displays the username, email and note but the response payload exposes the password hash.

Mitigation -

1. Make sure that the API endpoints only send the data that is required and not the entire object.

IDOR (Insecure Direct Object Reference)

- IDOR occurs when an application accepts a direct reference to an object (like a user ID in a query parameter) and fails to perform an authorization check to confirm the current user is allowed to access that specific object.
- In the website, a person logged in with his account can search for the other user by entering the user's user ID in the "explore users" input.
- The page shows only the username of the searched user, but due to EDE (Excessive Data exposure), one could easily have access to the hashed password and the private credentials of the searched user in the Network tab by inspecting the page, therefore compromising confidentiality.



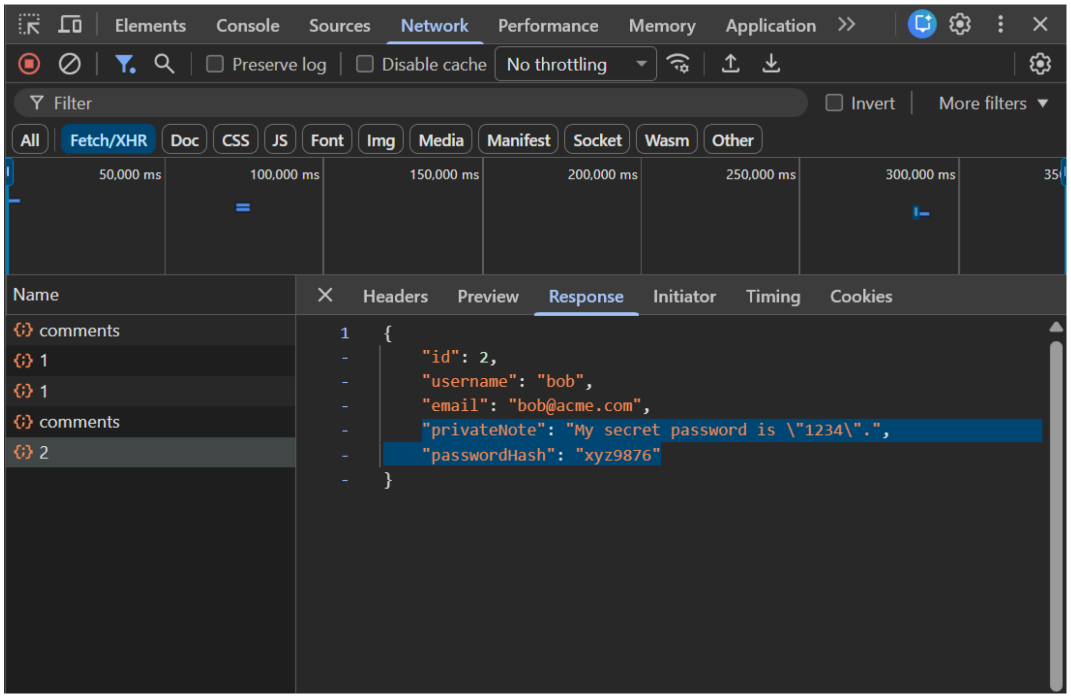
View Other Profile

2

View Profile

Username: bob

Fig.2. The DOM only displays the username



```
{
    "id": 2,
    "username": "bob",
    "email": "bob@acme.com",
    "privateNote": "My secret password is \"1234\".",
    "passwordHash": "xyz9876"
}
```

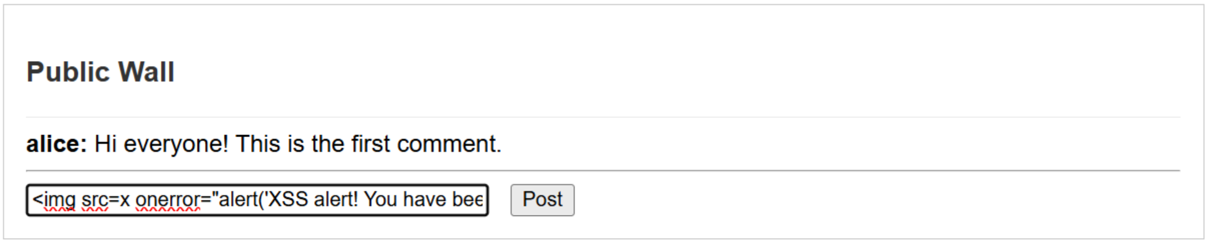
Fig.3. The Network tab shows the entire object data in the response payload

Mitigation -

1. In the backend verify that the person trying to change the data is the authorized user.
2. Check if the logged in user is the same as the one specified in the request.

XSS (Cross Site Scripting)

- Cross Site Scripting is the method in which an attacker can enter malicious script inside an input field and if there is no proper sanitization in the server and if the DOM simply adds the data as HTML Content, the script would run and exploit the other user accounts.
- In the website - We see this happen in the comments input. If a user enters a malicious element as input, it gets stored inside the database with no validation and gets displayed as a raw HTML element.


Public Wall

alice: Hi everyone! This is the first comment.

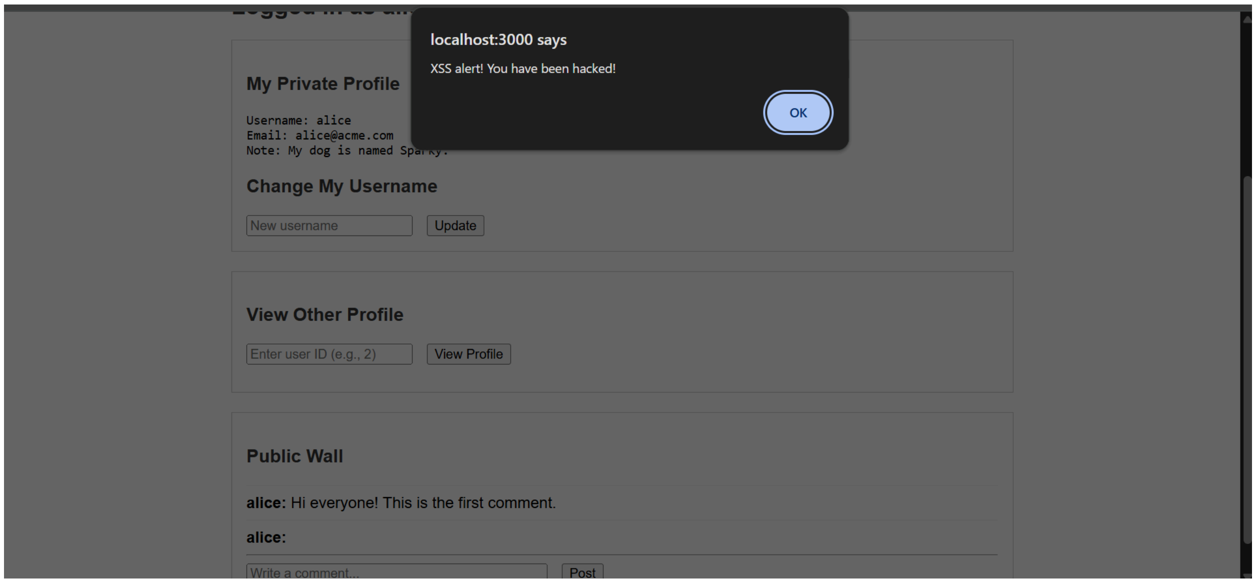
<img src=x onerror="alert('XSS alert! You have bee

Fig.4. Entering a malicious element inside the input

- As we see, we are entering a raw <img> element inside the input box.
- This element will trigger an error since src=x is not valid.
- The handler will run the script. In this case it is just an alert, but what if it is an actual dangerous script.
- As this input would get stored in the database, any user who opens the comments section will be a victim of the attack.

Mitigation -

1. Do not append the dynamic data directly using HTMLContent property.
2. Use textContent property to append information such as comments, text, messages etc.
3. Add proper sanitization in the backend before letting the user input interact with the database.


localhost:3000 says

XSS alert! You have been hacked!

OK

My Private Profile

```
Username: alice
Email: alice@acme.com
Note: My dog is named Sparky.
```

Change My Username

New username | Update

View Other Profile

Enter user ID (e.g., 2) | View Profile

Public Wall

alice: Hi everyone! This is the first comment.

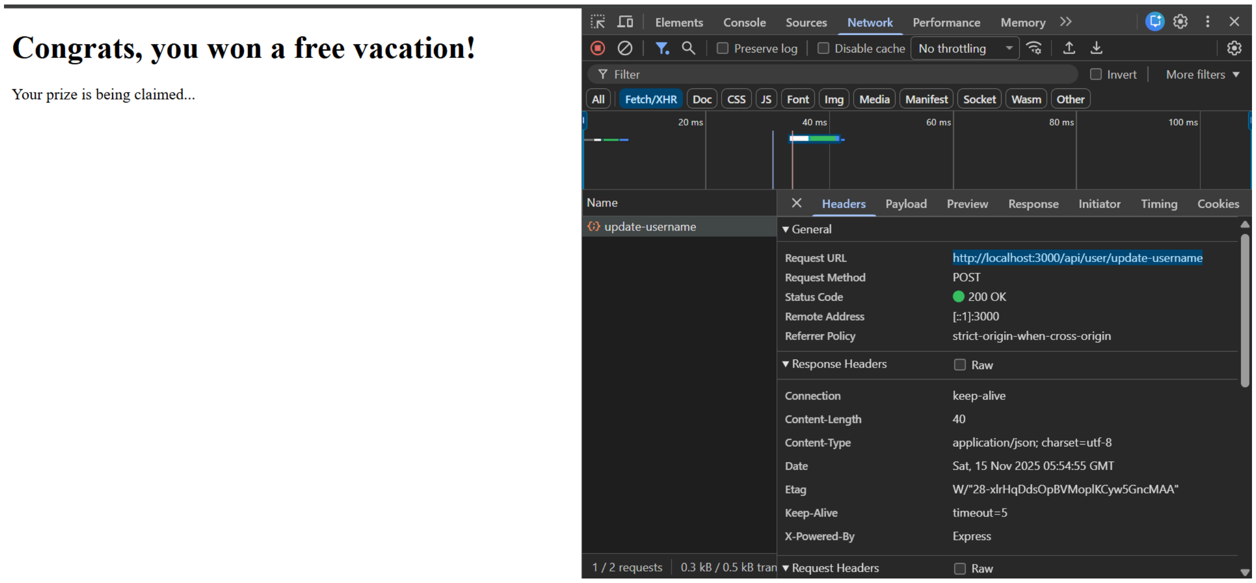
alice:

Write a comment... | Post

Fig.4. XSS Execution

CSRF (Cross Site Request Forgery)

- In a CSRF attack, the attacker generally tricks the user to make a request to the server without the user knowing about the request.
- The server authenticates the user since the browser sends the session cookies along with the request.
- However, the evil.html page will not be able to access the response due to Cross Origin Policy blocking the access to the response. But the browser sends the request and the server completes the action.
- In this website - We created an evil.html page that ran a script to modify the user’s username when the user opens the page. We notice that the script makes a call to the server and the server validates it since the browser also sends the session attached with the user along with the request and hence, the server authenticates the request.

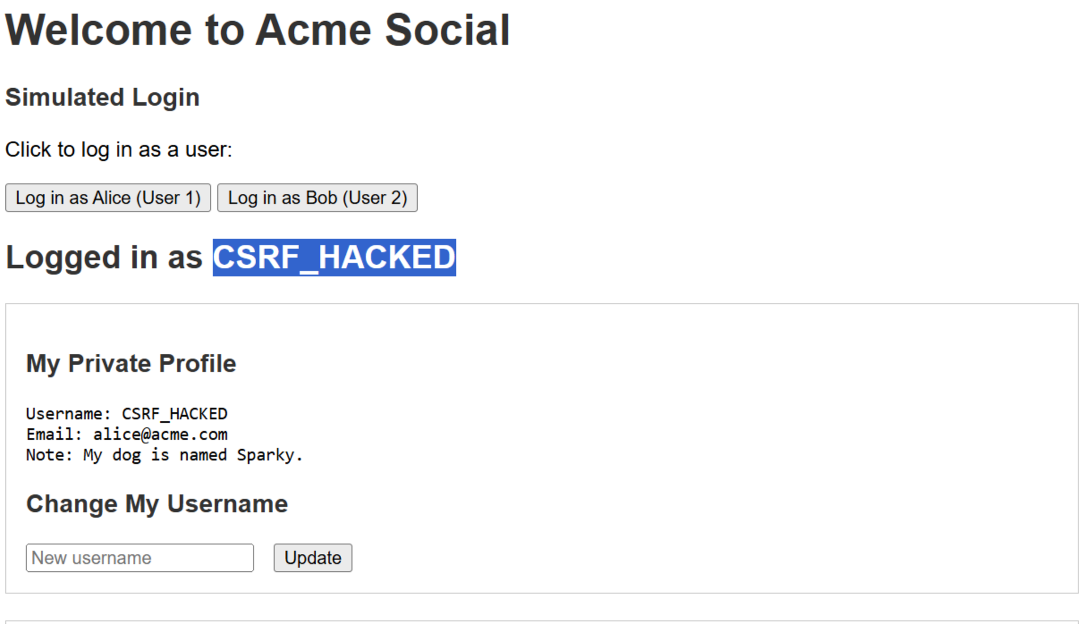


Congrats, you won a free vacation!

Your prize is being claimed...

```
update-username

General
Request URL        http://localhost:3000/api/user/update-username
Request Method     POST
Status Code        200 OK
Remote Address     [::1]:3000
Referrer Policy    strict-origin-when-cross-origin

Response Headers
Connection         keep-alive
Content-Length     40
Content-Type       application/json; charset=utf-8
Date               Sat, 15 Nov 2025 05:54:55 GMT
Etag               W/"28-xlrHqDdsOpBVMoplKCyw5GncMAA"
Keep-Alive         timeout=5
X-Powered-By       Express

Request Headers

1 / 2 requests | 0.3 kB / 0.5 kB tran
```

Fig.5. evil.html page that makes a request without the user knowing about it



Welcome to Acme Social

Simulated Login

Click to log in as a user:

Logged in as CSRF_HACKED

My Private Profile

```
Username: CSRF_HACKED
Email: alice@acme.com
Note: My dog is named Sparky.
```

Change My Username

Fig.5. The CSRF attack has been executed and the user credentials have been changed

Mitigation -

1. Use the anti-CSRF middleware to restrict the CSRF attack. This middleware attaches a unique, one-time use CSRF token along with the HTML page.