IDATT2503 - Exercise 1 - Edvard Berdal Eek

Task 1:

A1 Broken Access Control:

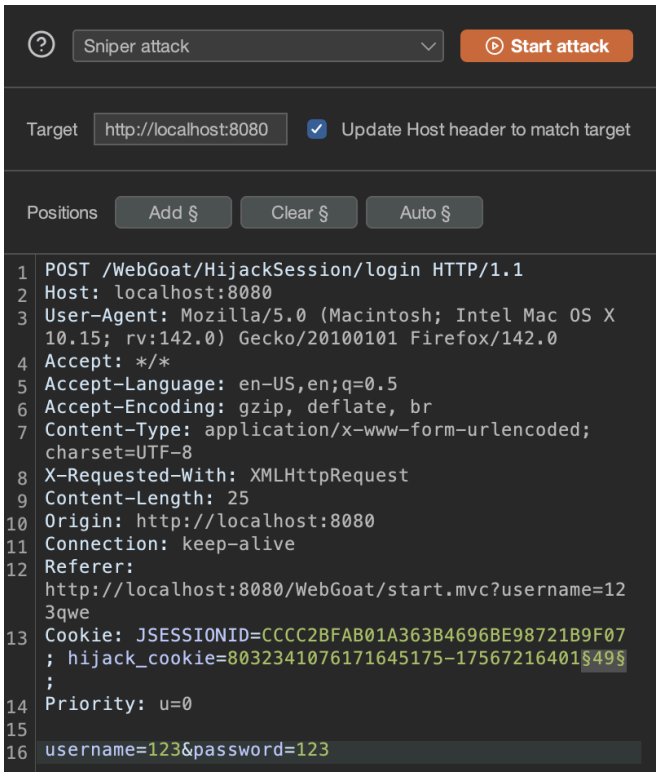
- Hijack a session

After inspecting the hijack_cookies in the responses from several login requests, i notices that the first part of the cookie usually increments by one and the, while the second part is seemingly affected by the time.

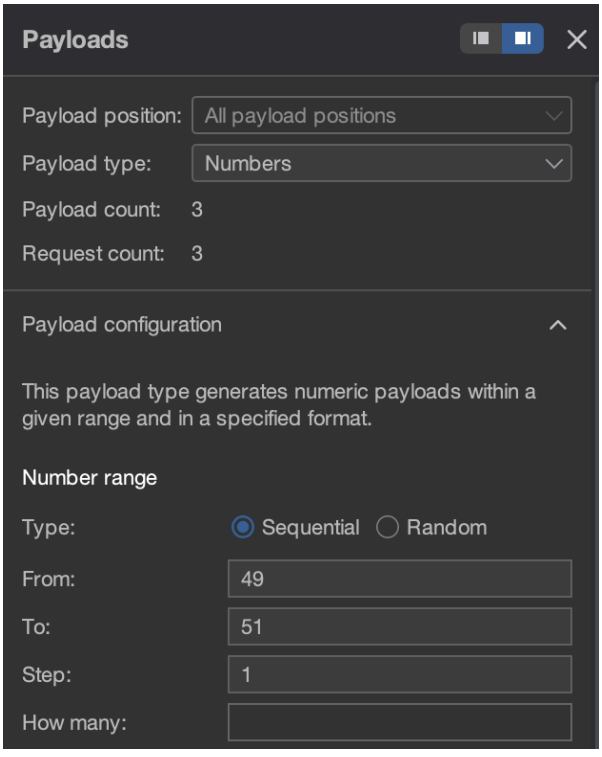
I then attempted to quickly send multiple requests looking for any increments to the first part that were larger than one, which could suggest that the one inbetween is already in use. I eventually found an increment of two, however the second part had a very large range. I then used the Sequencer tool in Burp Suite which allowed me to send login requests and capture responses from very quickly. I sorted the saved tokens and found two tokens where the first part was incremented by two and the second part had a small range:

80323410761716451**74**-1756721640149 and 80323410761716451**76**-1756721640151

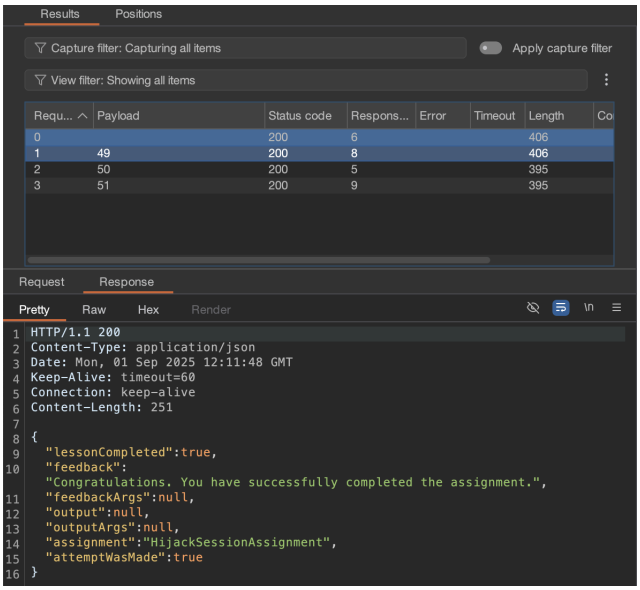
I then put a login request in the Burp Suite intruder tab.



This tab allowed me to brute force with requests containing the hijack token set with the suffix on the first part between the other tokens, 75, and within a range from 49 to 51 for the second part.

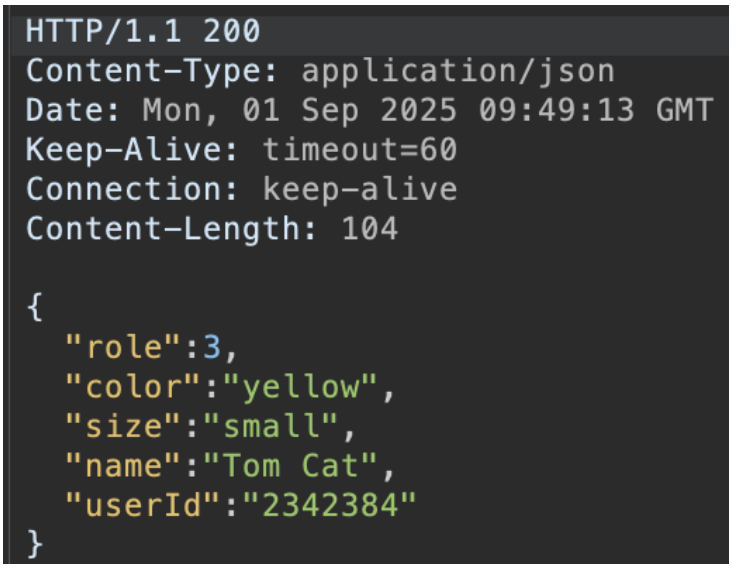


Which led me to finding the correct hijack token.



- Insecure Direct Object References

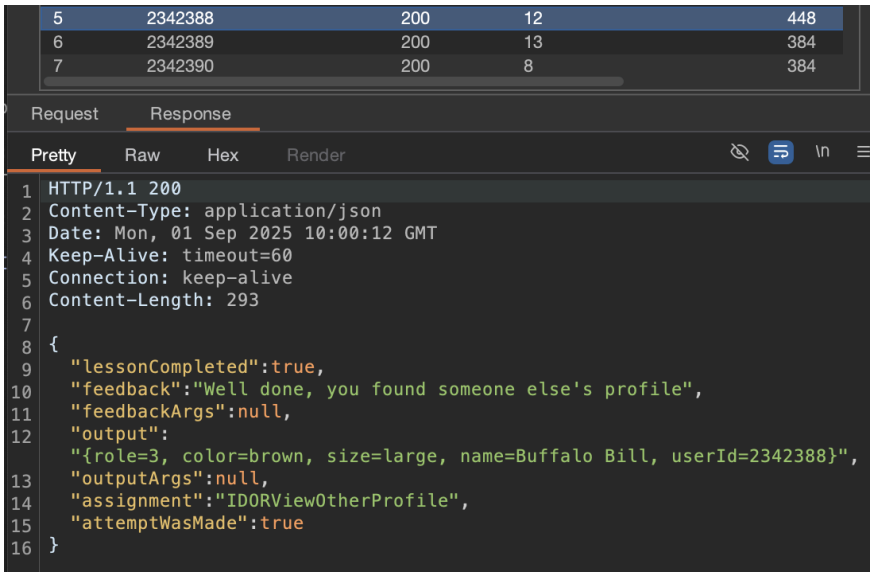
In the secod step the response from the view profile request contained to additional fields, role and userId:



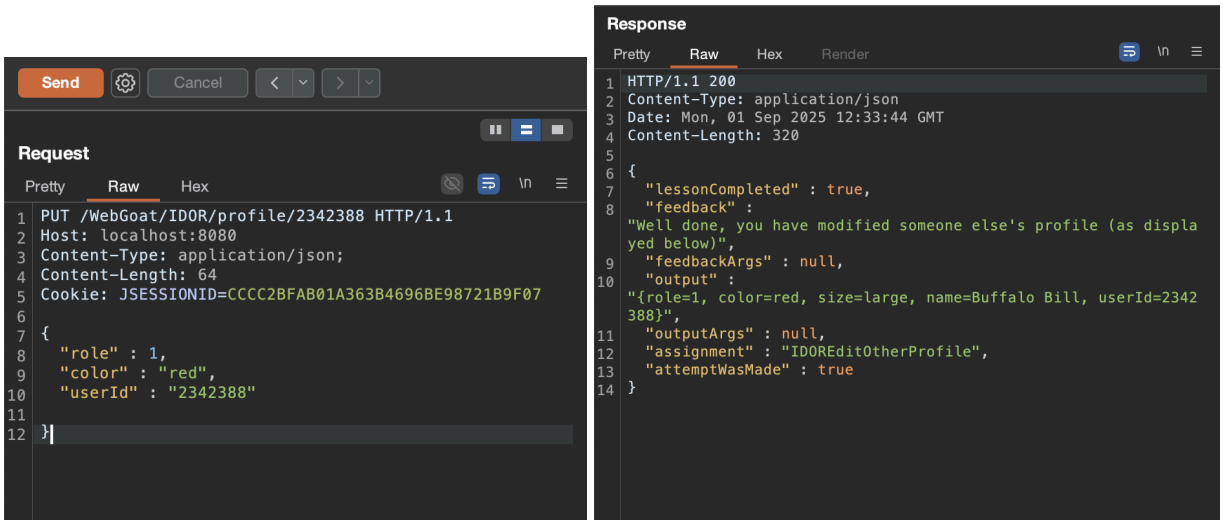
Looking at the path in the previous request i predicted that the view profile path would be the same, but with an additional user id: WebGoat/IDOR/profile/2342384.

In the last step i assumed that there would be another userId within the nearest hundred values. I therefore sent the original view profile request to the Burp Suite intruder tab and attempted with incremented userId values.

Which made me find the valid userId 2342388.



In the repeater i constructed a PUT-request to the url containing the newly found userId, altering the role and color. I had to test a bit before finding out what headers and content were necessary for the task to succeed.



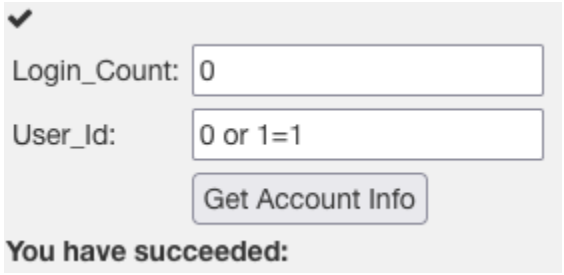
A3 Injection:

- SQL Injection (intro)

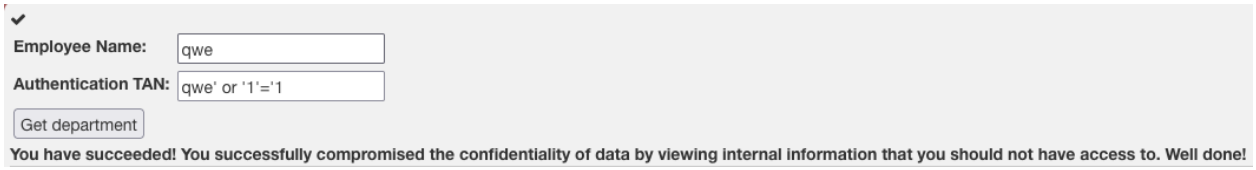
I felt it was unnecessary to include pictures from task 2-5, but image above proves completion.

9)

10)



11)



12)

Employee Name:

asd

Authentication TAN:

asd'; UPDATE employees SET salary='999999999' WHERE auth_tan='3SL99A

13)

Action contains:

*'; DROP TABLE access_log; SELECT * FROM employees WHERE '1'='1

- Path traversal

In part 2 i first attempted to edit filename of the image in the request which did not work in here. I then just attempted to travers in the input of the full name which solved the part.

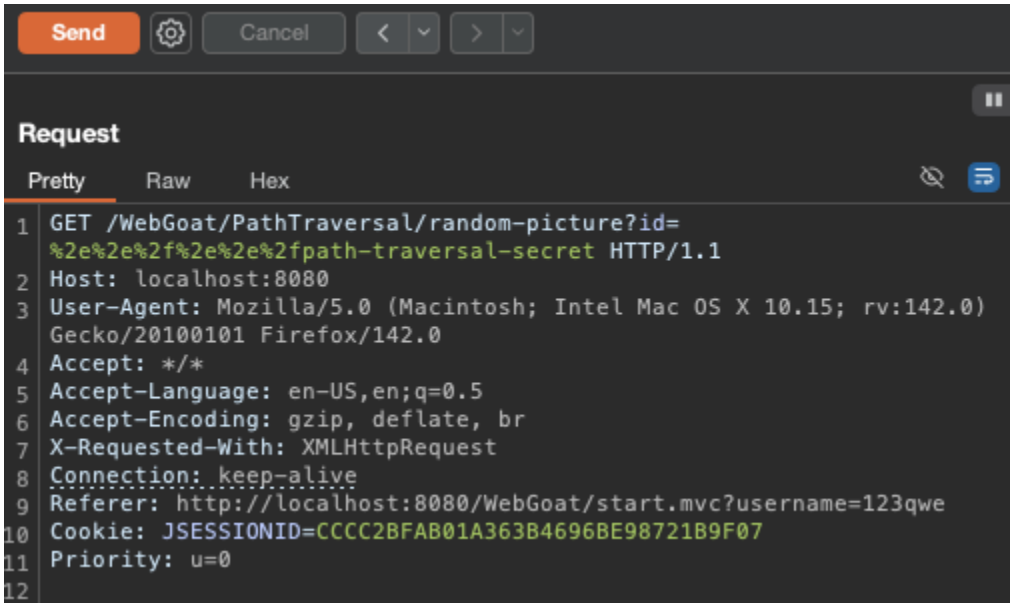
Part 3 i intertwined two traversals '../' leading to a total prefix of '..../'. Presumably the programmer tried to patch the issue by just removing a '../' if it exists.

Part 4 was the solution i attempted in the first task where i altered the request to contain traversed prefixes in the filename of the image sent.

```
------WebKitFormBoundaryAdHng5dBz53kPd8L
Content-Disposition: form-data; name="uploadedFileFix";
filename="../../NobodyWantsToWork.png"
Content-Type: image/png
```

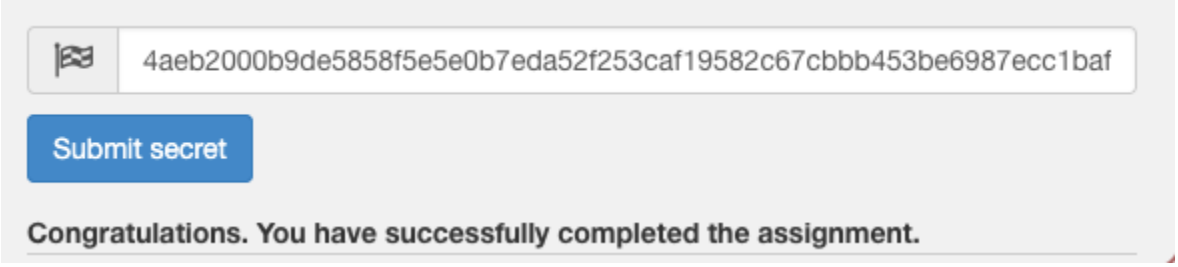
Part 5 of this task led to alot of problems. For instance, the WebGoat application was no longer accessible until i disabled the proxy i had set up in FireFox for burp suite at port 8081. After asking the teaching assistant i was told to rather use the Open in browser functionality in Burp Suite so i wouldnt have to use a proxy.

My solution was the following:

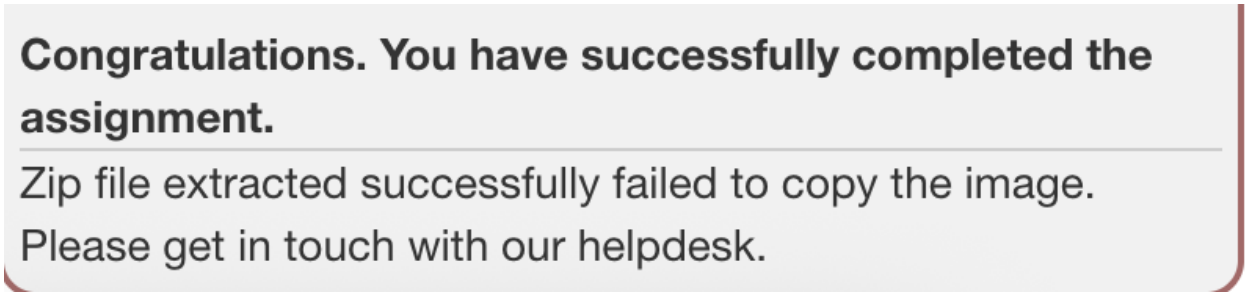


Using the URL-encoded version of '../../' with the file name appended.

In addition, when i did disable it and provided the secret, the SHA-512 hash of my username, it gave me the completion without registering the assignment as completed in the colored tabs.



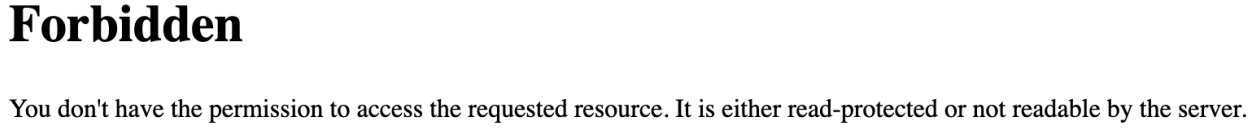
And in part 7 i am advised to contact the helpdesk as the zip was extracted successfully but it fails to copy the file.



Task 2:

- Insecure Direct Object References

Creating a page and moving through differenct pages i noticed that the pages were directly referenced in the URL with the format: **page/pageId**. I then attempted changing the page id number. This lead me to discover that page 4 was forbidden.



I also noticed that when editing a page the URL had this format: **page/edit/pageId**. I then attempted to edit the forbidden page which lead me to the first flag.

- Cross-site scripting (XSS)

The page claims that scripts are not supported. I first attempted this in the body of the page, which ended up being ‘scrubbed’. But editing the page name to contain a script allowed lead to another flag

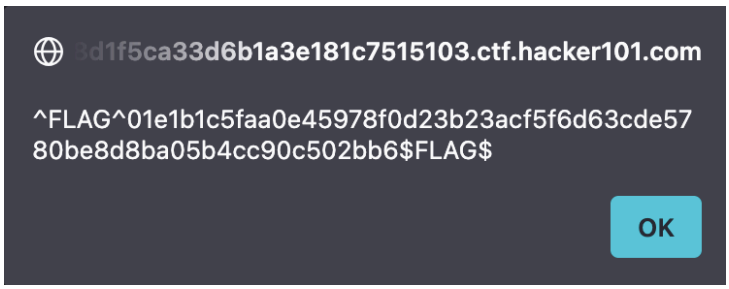
<-- Go Home

Edit Page

Title: <script>lol</script>

THIS WAS MY PAGE ONCE

Save



<-- Go Home

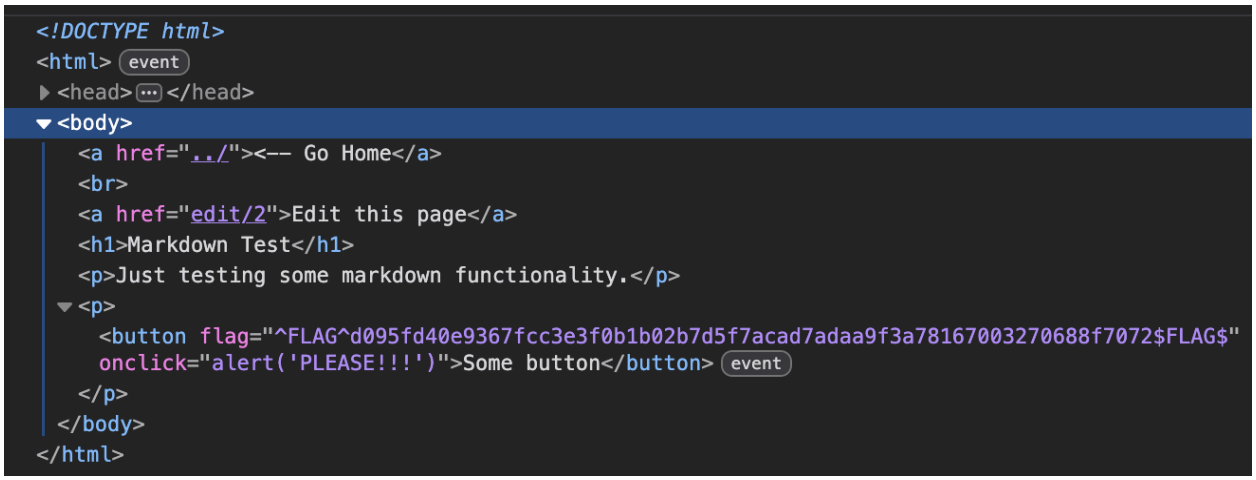
Edit Page

Title: Markdown Test

Just testing some markdown functionality.

<button onclick=alert('PLEASE!!!')>Some button</button>

After clicking the button an alert appeared when i then inspected the page it now revealed another flag!



- SQL Injection

^FLAG^f51d544f06107a2907e71f89bfca60799d31de68f1670159130ba1db0317f27a$FLAG$