Easy-Web_challenge

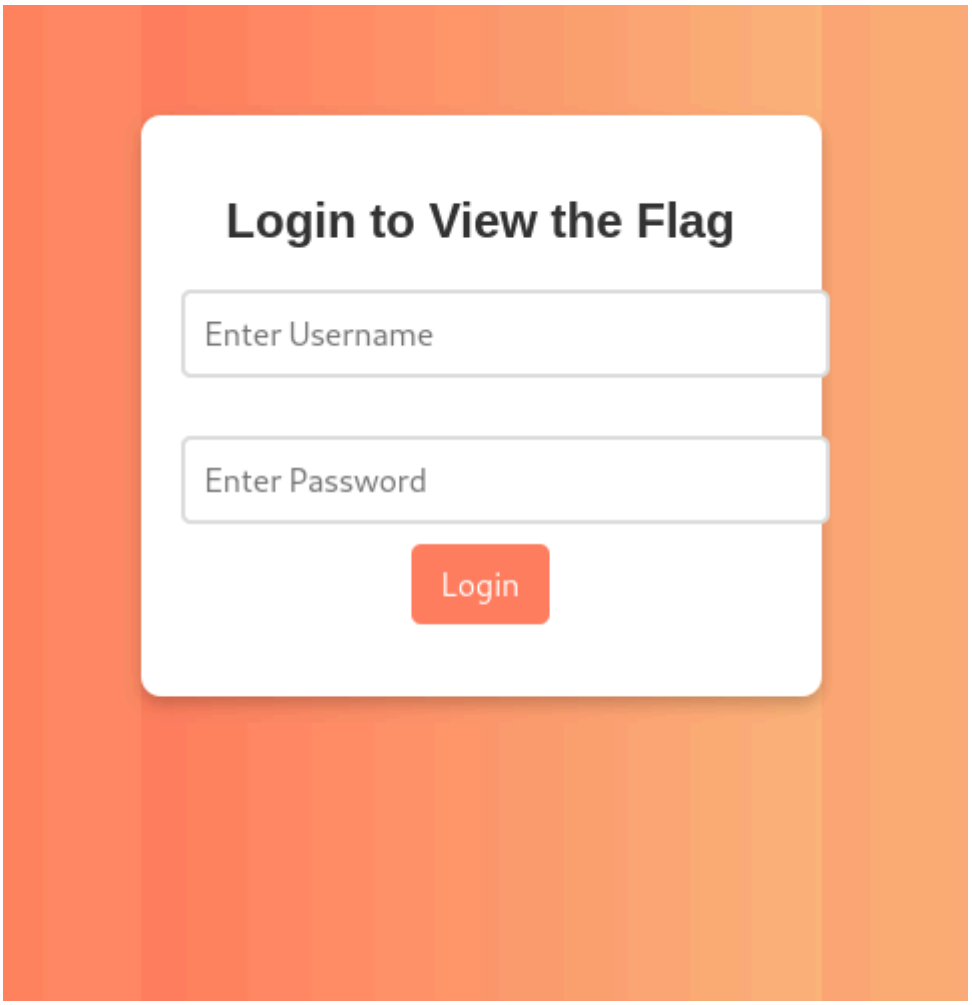
Description : **To Login This Page And Get Flag**

SOLUTION:

- **Navigate to the challenge URL: https://web-chall-ten.vercel.app/.**
- **A login page prompts for a username and password.**

Initial Analysis

- **Observed no immediate points of interest on the login page.**



- **Opened Firefox Developer Tools to analyze the source code.**

Inspecting the HTML Source

- **Found that a JavaScript function `checkCredentials()` is being called.**

```html
<!DOCTYPE html>
<html lang="en"> flex
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CTF Web Challenge</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body> flex
    <div class="container">
        <h2>Login to View the Flag</h2>
        <input id="username" type="text" placeholder="Enter Username">
        <br>
        <br>
        <input id="password" type="password" placeholder="Enter Password">
        <button onclick="checkCredentials()">Login</button> event
        <p id="flag"></p>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

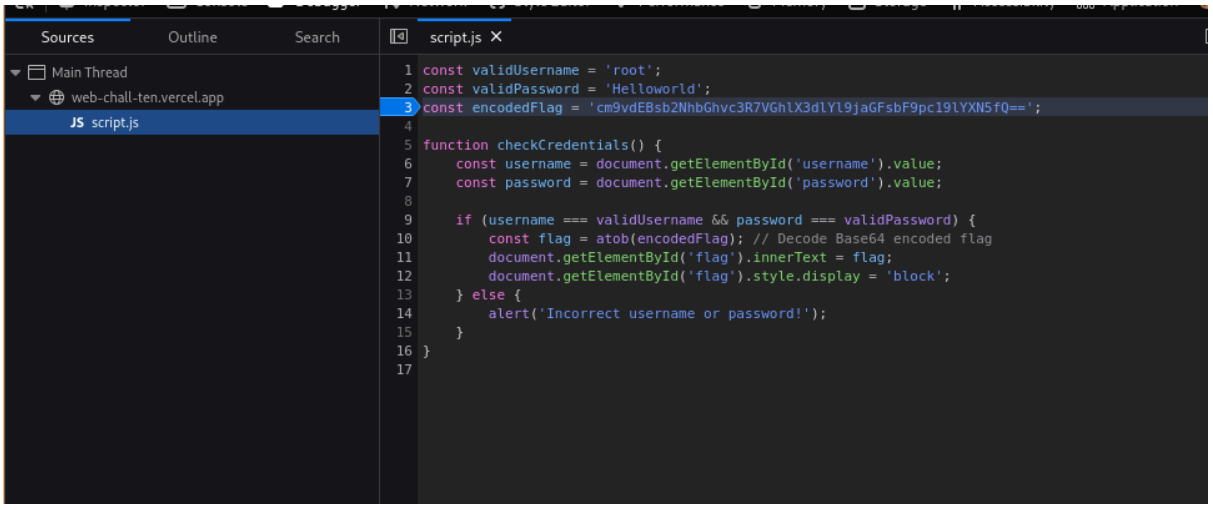
- **The function is referenced in `script.js`.**

Examining `script.js`

- **Opened the Debugger tab in Developer Tools.**
- **Navigated to and opened the `script.js` file.**

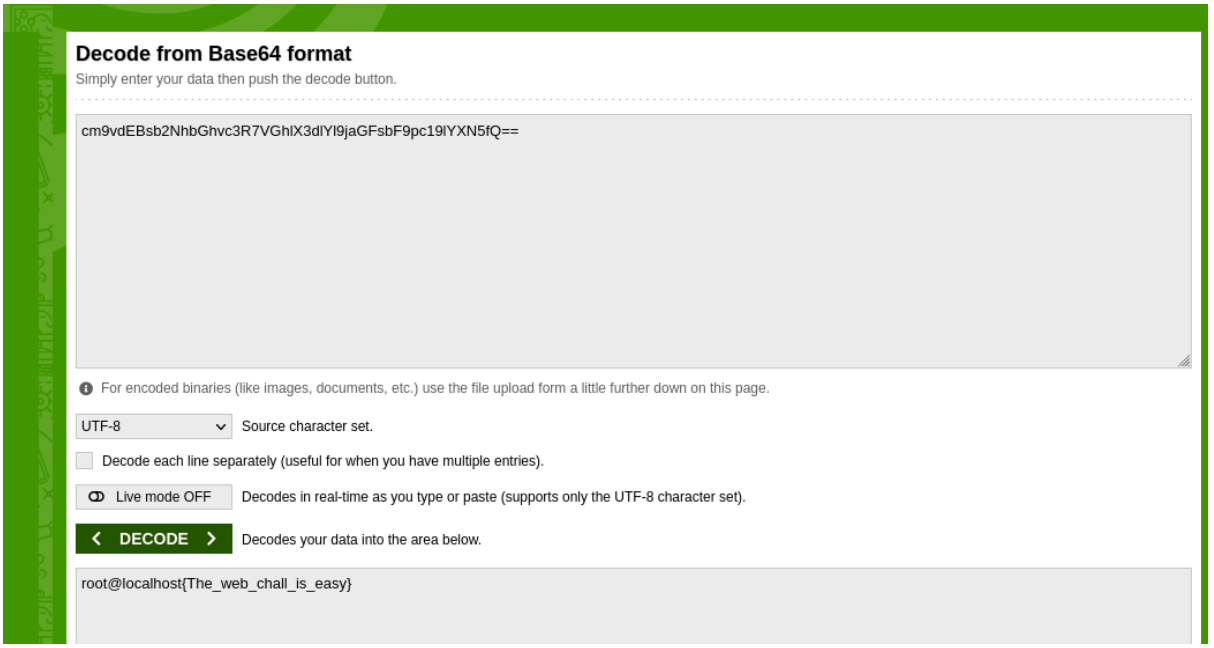
Finding the Flag

- **Discovered that the flag is base64-encoded within the `script.js` file.**



Decoding the Flag

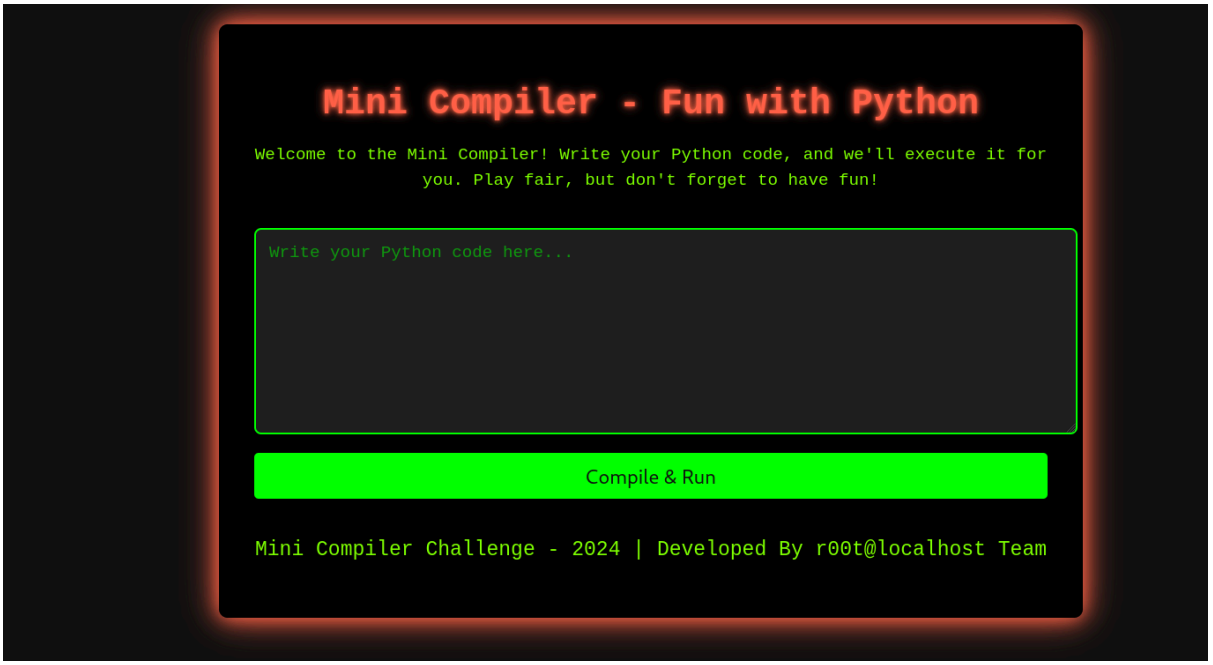
- **Copied the base64-encoded string and decoded it to retrieve the flag.**



Mini Vulnerable Compiler

Description : In this challenge, you have access to a simple online compiler that executes Python code. The code you submit is run on the server, and your goal is to exploit this vulnerability to retrieve the secret flag

SOLUTION:

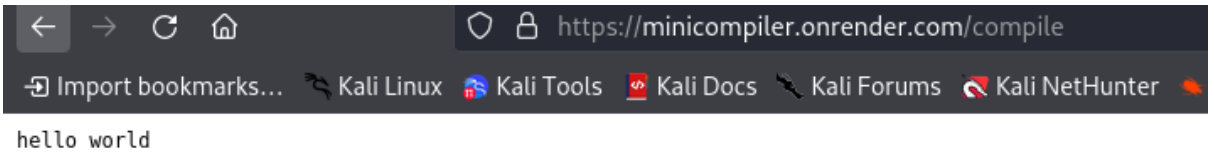


Accessing the Website

- Visited the target website, which provided an input field that appeared to support Python code execution.

Testing Code Execution

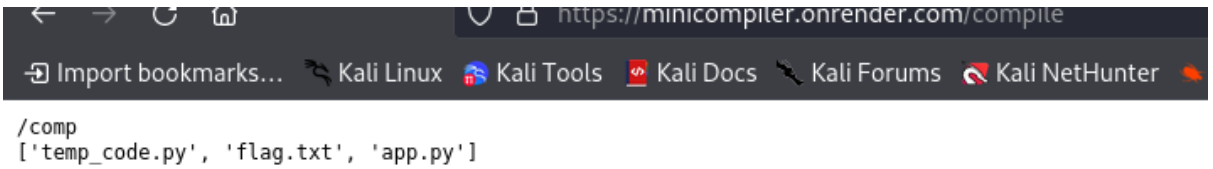
- Entered a simple Python command: `print('hello world')`.



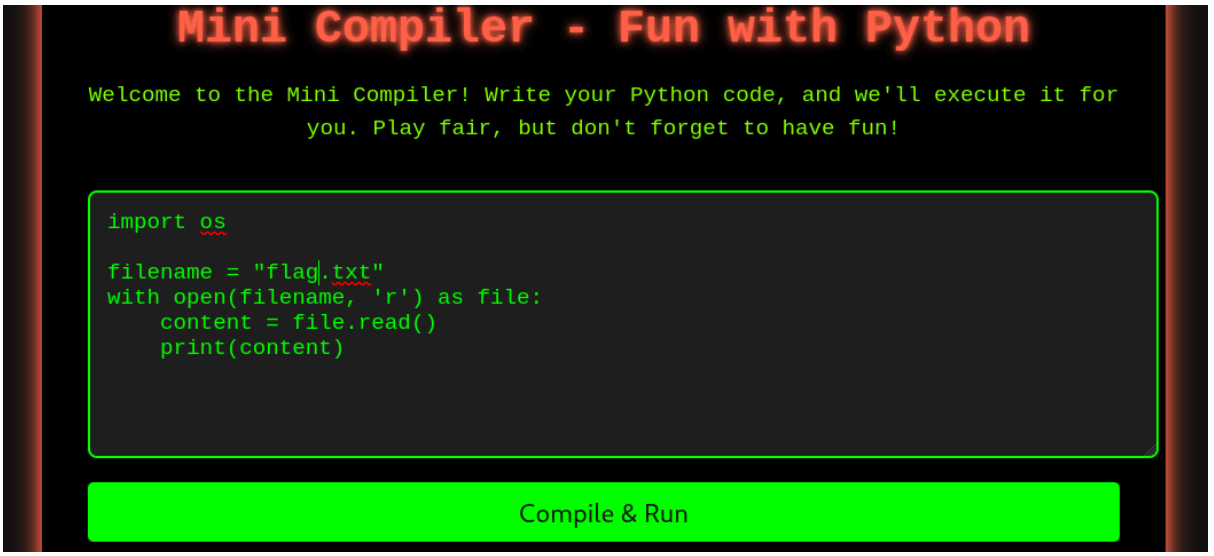
- Verified successful execution as the output displayed "hello world".

Expanding Reconnaissance

- Leveraged ChatGPT to craft Python commands for further reconnaissance on the system.
- 



Locating the Flag

- Discovered a file named `flag.txt` in the current directory.
- 

- Confirmed that the file was readable using Python commands (with ChatGPT's help).

Retrieving the Flag

- Successfully read the content of `flag.txt` to reveal the flag.
- 

iDoor: The Secret Portal

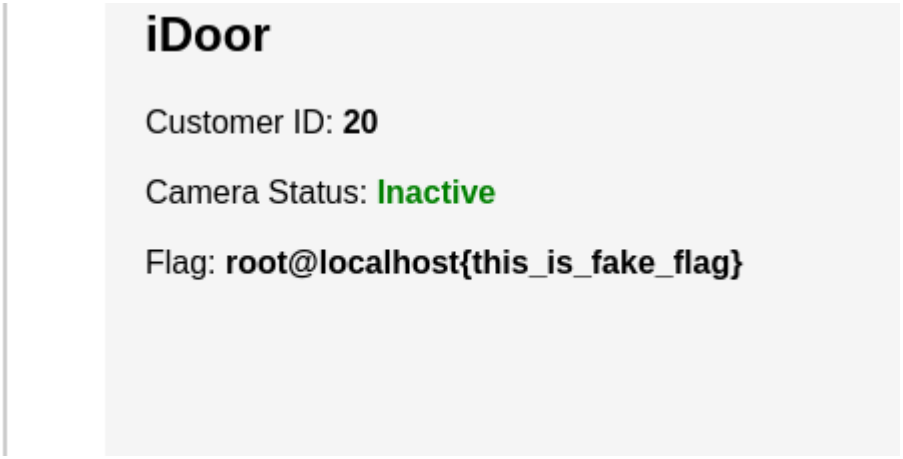
Description : The 'iDoor' web challenge presents a secure access system with an interface that resembles a CCTF camera page. It tests your skills in web exploitation and security analysis.

HINT: sha256

Solution:

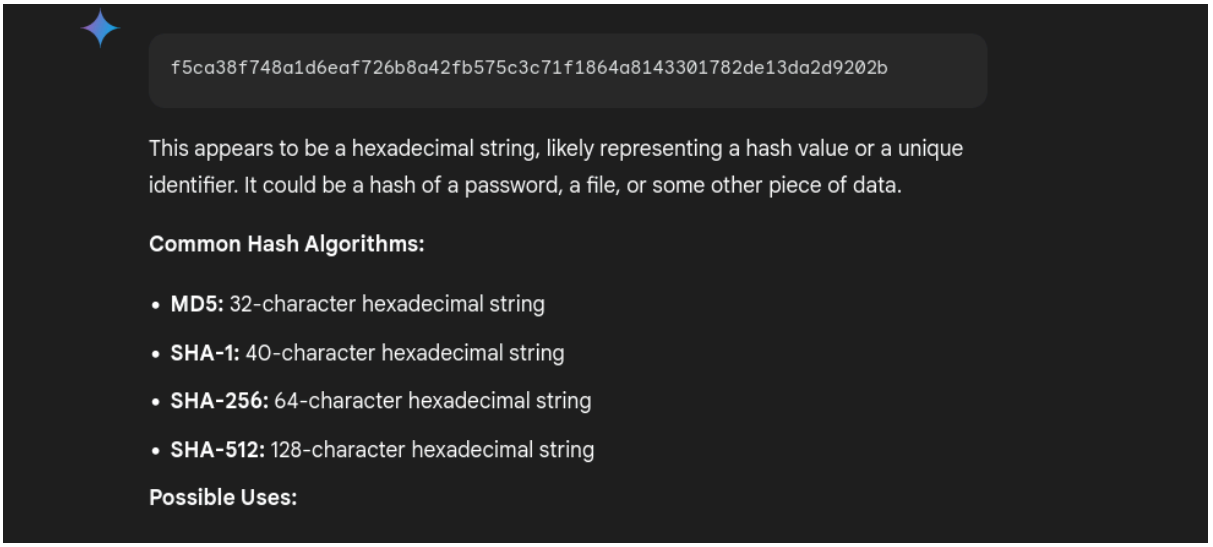
Initial Analysis

- **Started by analyzing the webpage's source code (a standard first step in web CTF challenges).**

- **Found nothing particularly interesting except a fake flag.**
- 

Spotting the Key Parameter

- **Noticed a parameter named `camera` in the URL, holding a random junk value.**
- **Copied the value and ran it through Gemini, identifying it as some kind of hash.**

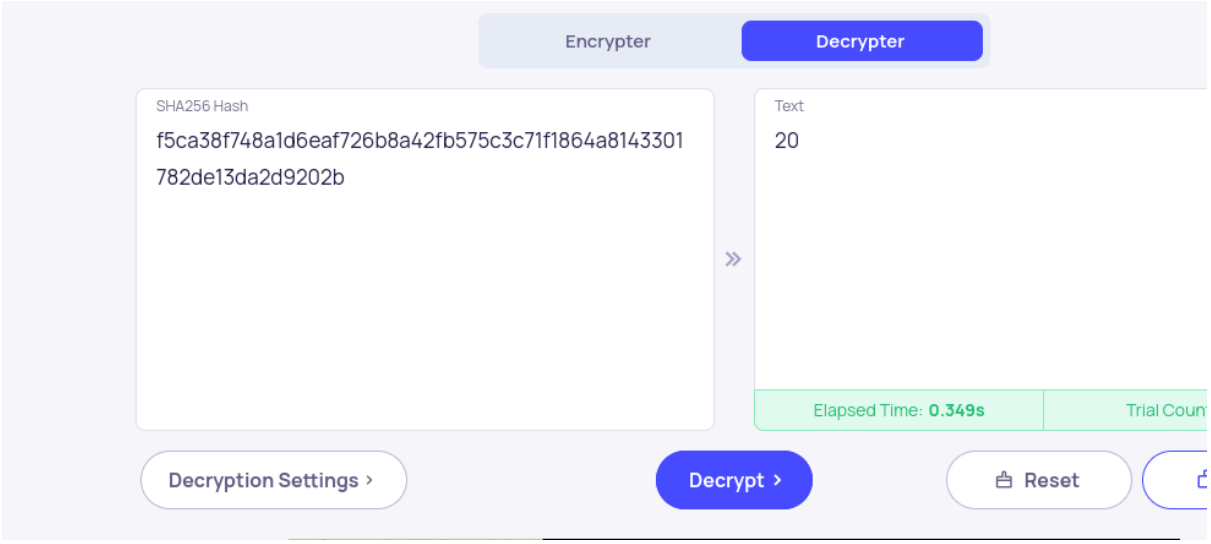


Recognizing SHA256

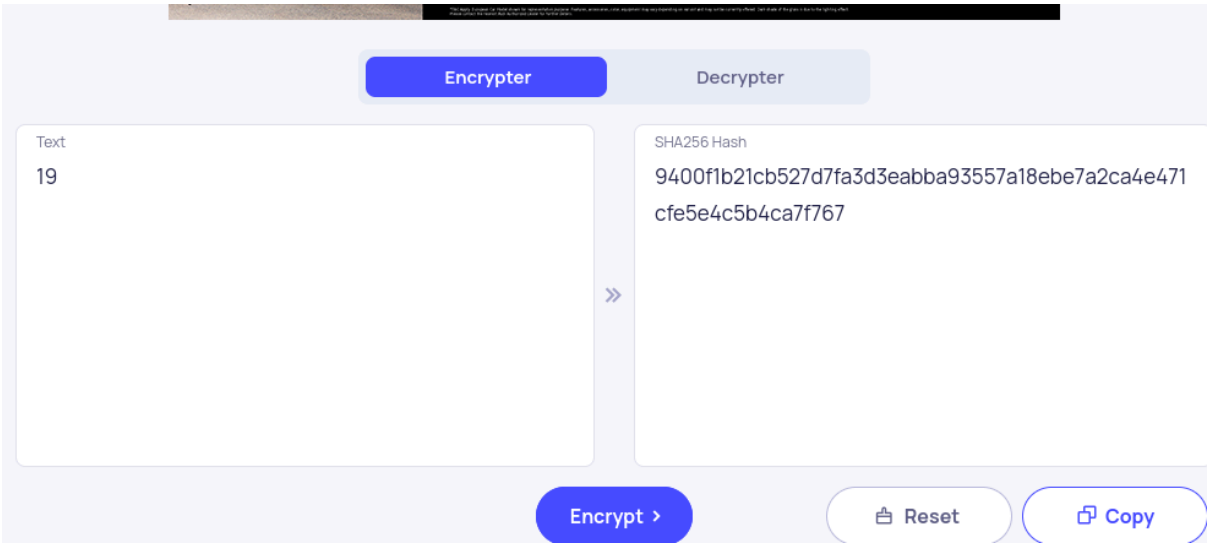
- **With the provided hash list from Gemini and the challenge hint mentioning SHA256, deduced the hash was likely a SHA256 hash.**

Cracking the Hash

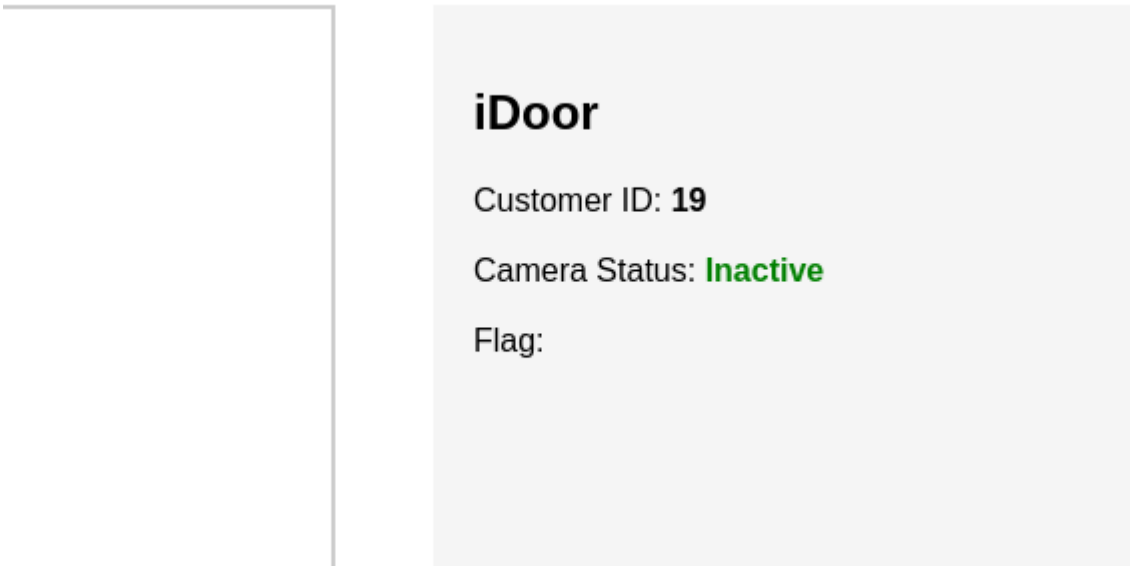
- **Searched fpr online SHA256 cracker and found: 10015.io/tools/sha256-encrypt-decrypt.**
- **Successfully cracked the hash to reveal the value `20`, which matched the `customer_id` on the website.**

- 

Experimenting with Parameter Values

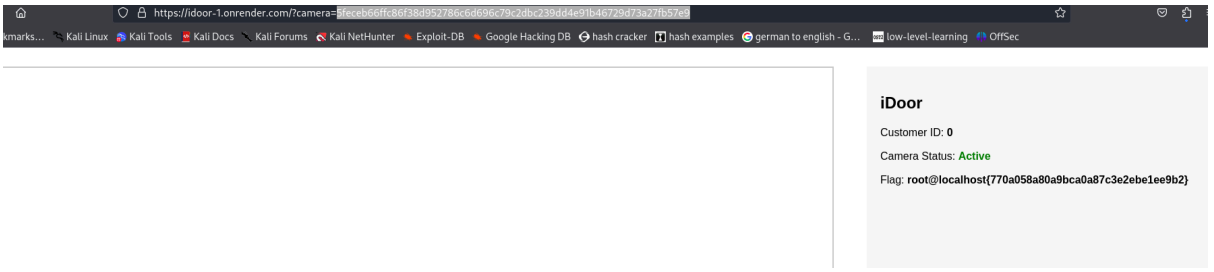
- **Changed the `camera` parameter value to a SHA256 hash of `19` and updated the URL.**
- 

- **Observed that the fake flag was removed for `customer_id` 20, suggesting the flag was tied to specific `customer_id` values.**



Finding the Real Flag

- **Tested various `customer_id` values and discovered that `customer_id = 0` revealed the original flag.**

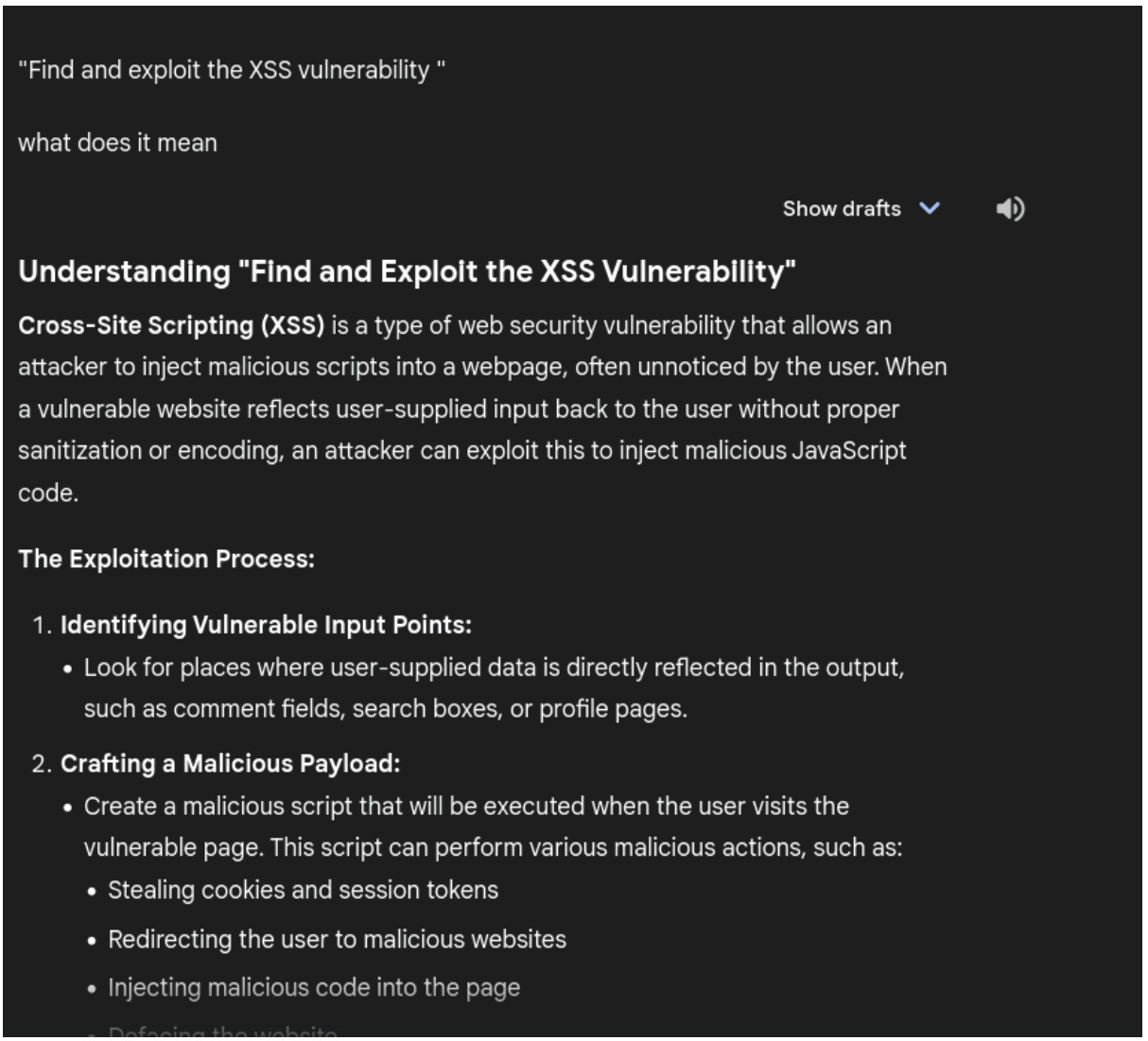


XSS vulnerability

Description : "Find and exploit the XSS vulnerability "
Solution :

Understanding XSS

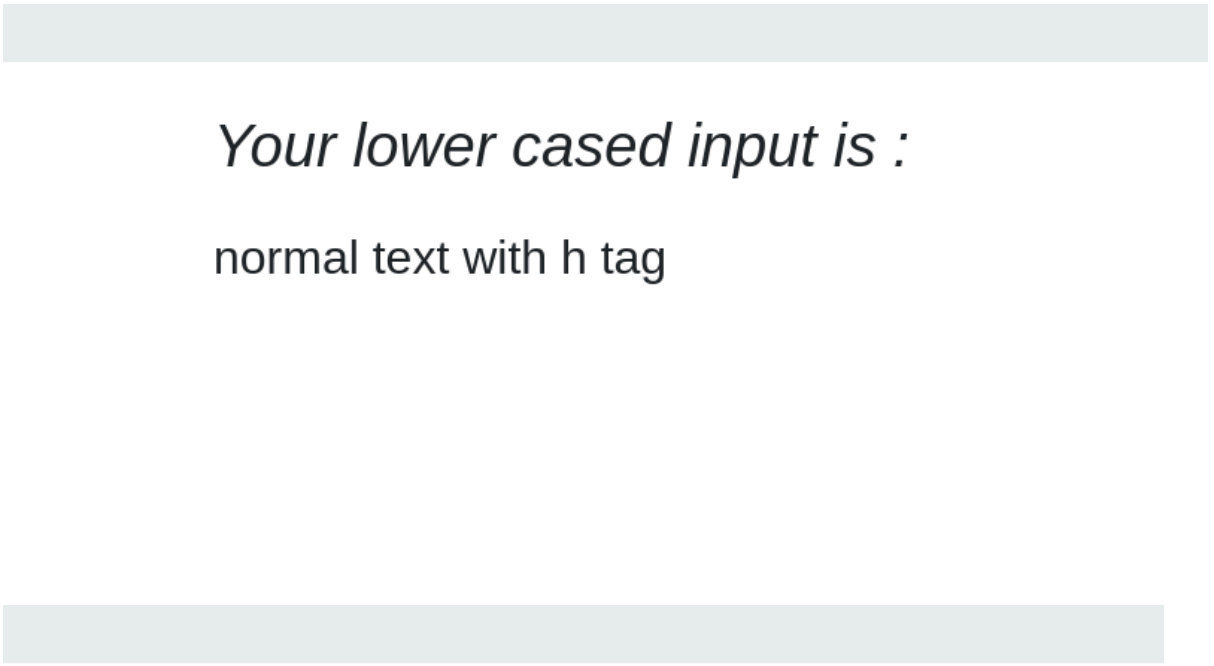
- **Used Gemini to reinforce my understanding of XSS by simply prompting it using the challenge description.**

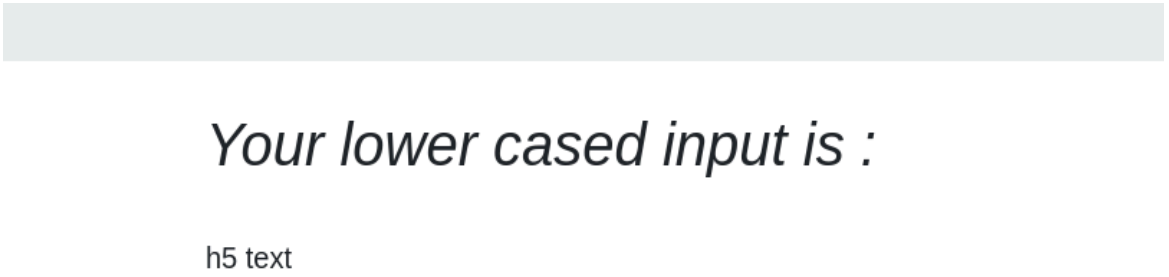
- 

- **Learned that XSS (Cross-Site Scripting) is a vulnerability where unsanitized user input can lead to malicious HTML/JavaScript execution, causing serious consequences.**

Testing for Vulnerability

- **Tested the input field by entering a basic `<h5>` tag to observe changes in output formatting.**

- 

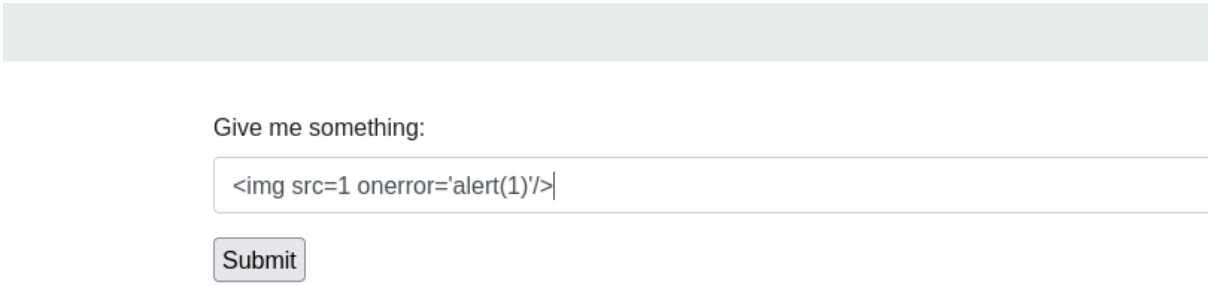


-
- **Confirmed the input field was vulnerable as the output text size was smaller, indicating the tag was processed.**

Crafting a Malicious Payload

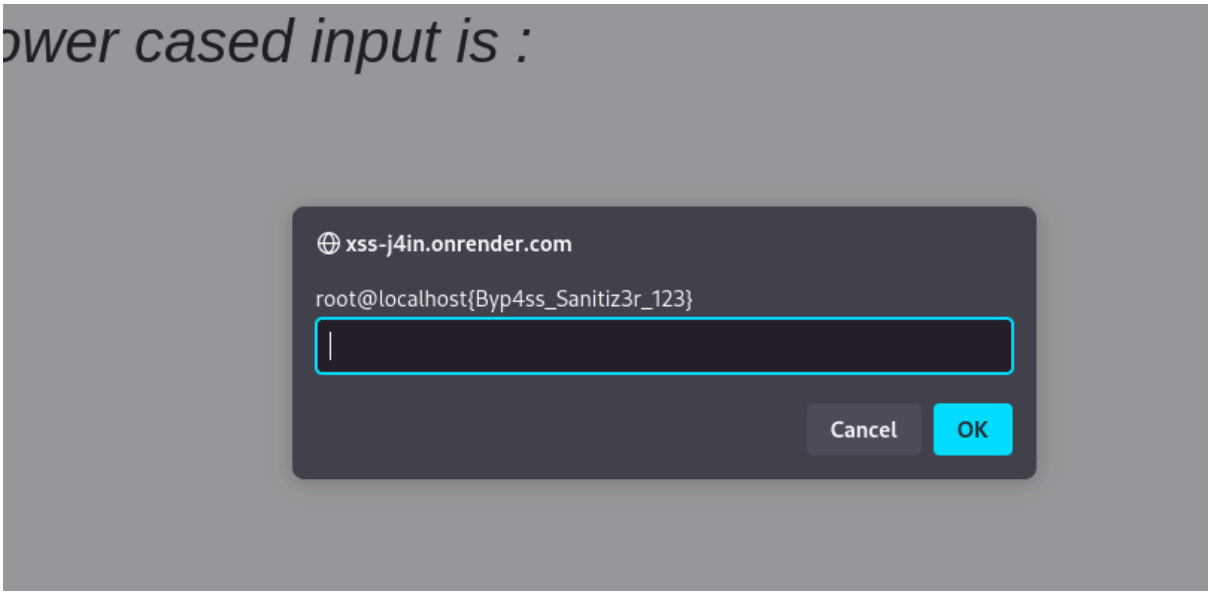
- **With guidance from Gemini, crafted a simple XSS payload:**

  ****



Retrieving the Flag

- **The malicious input successfully triggered an alert box, displaying the flag.**

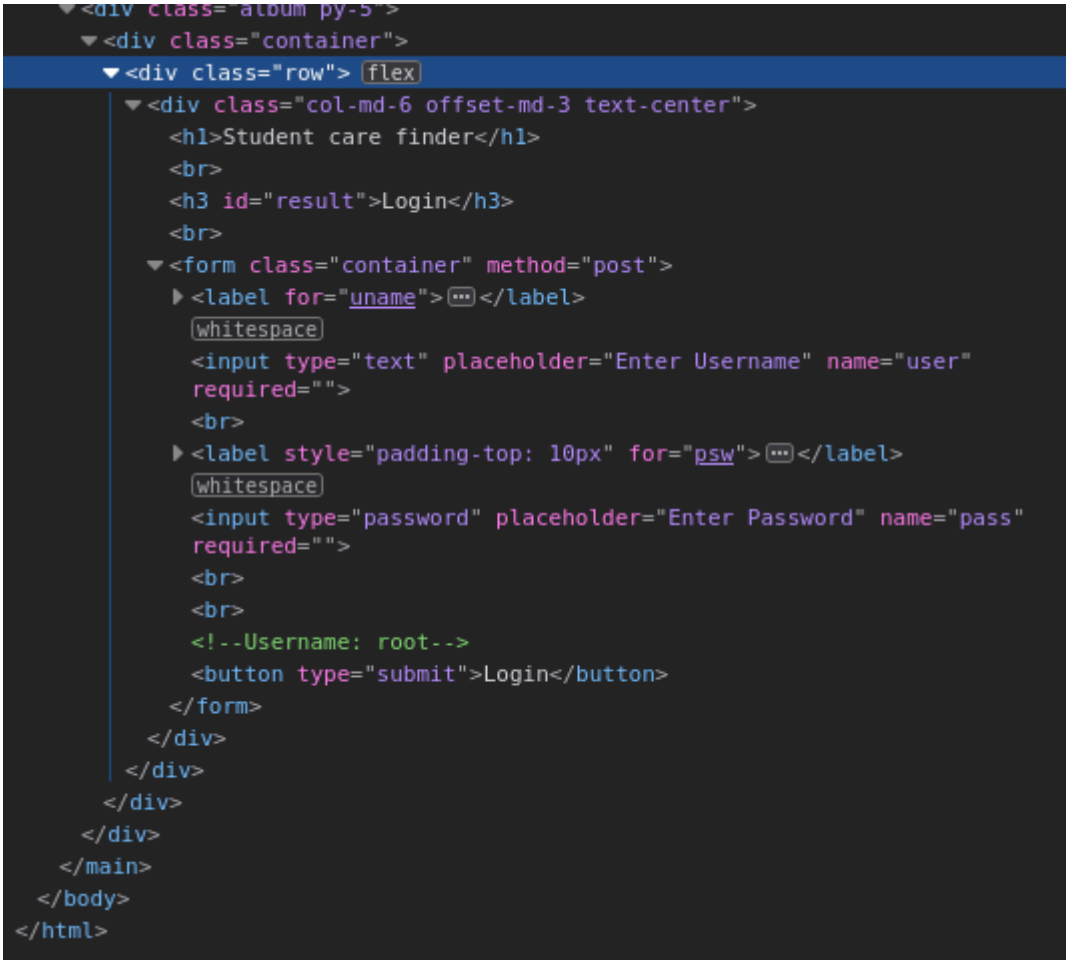


jwt

Description: You've logged into a web app with `demo:demo`, but it's got more holes than Swiss cheese. Your job: find a way to exploit its weak security, escalate your privileges, and sneak into restricted areas. Can you prove the app's defenses are a joke?

Solution:

Analyzing the HTML Source

- **The first step was to analyze the HTML source code of the challenge. A hint in the source revealed that the username to test was `root`.**
- 

Understanding JWTs

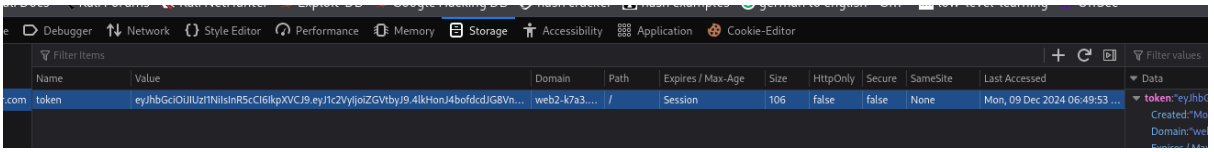
- **JWT (JSON Web Token) is a widely used component in authentication and cookie management for web applications.**
- **Although I understood JWTs conceptually, I wasn't confident in identifying and exploiting vulnerabilities related to them.**

Learning About JWT Vulnerabilities

- **To bridge the gap, I referred to an article on PortSwigger's site: What are JWTs?.**
- **The article highlighted that a common vulnerability in JWT-based challenges is the use of weak signing keys.**

- I also explored PortSwigger's lab: [JWT Authentication Bypass via Weak Signing Key] for practical insights into using tools like `hashcat` to brute-force signing keys.

Extracting the JWT

- Logged in to the application using the credentials `demo:demo`.
- 

- Captured the JWT token from the response and saved it as a `.txt` file to use it later for brute-forcing with `hashcat`.

Cracking the Signing Key

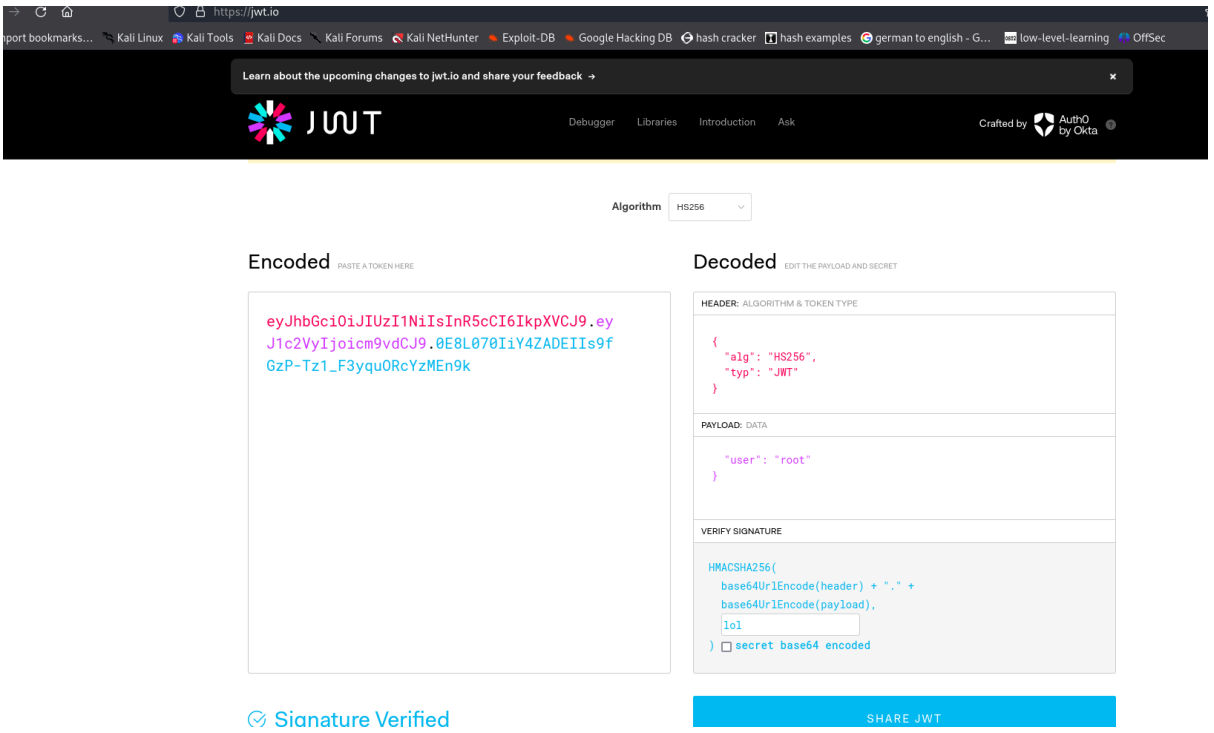
- Used `hashcat` to perform a brute-force attack on the saved JWT.

  ```
  hashcat -a 0 -m 16500 --show "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiZGVtbyJ9.4lkHonJ4bofdcdJG8VnrnzGL9KKUQwknSTccjAj-0_M" /usr/share/wordlists/rockyou.txt
  ```

- After some time, `hashcat` successfully cracked the secret signing key.

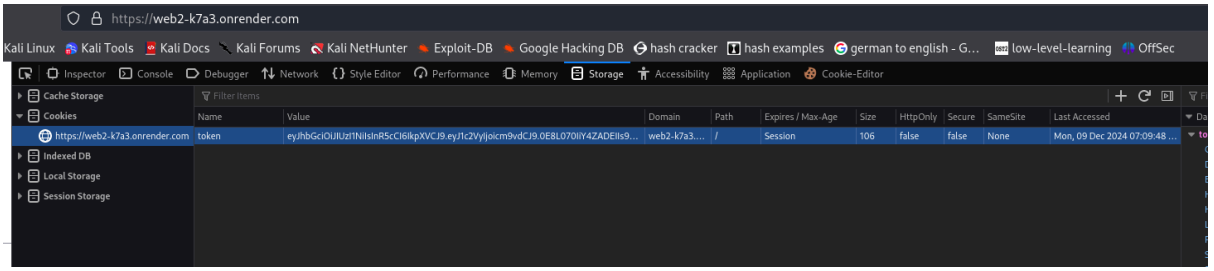
Manipulating the JWT

- Visited [jwt.io], a tool for decoding, encoding, and signing JWT tokens.
- Used the cracked signing key to create a new, signed JWT.
- This token was modified to set the username as `root`.



Injecting the Modified JWT

- **Opened the browser's Developer Tools and navigated to the Storage tab.**
- **Located the JWT in the cookies under the name `name`.**
- **Replaced the existing token with the newly signed JWT.**



Successfully Gaining Access

- **Reloaded the page, and the application granted access as the `root` user.**