

WEB CHALLENGES

Easy-Web_challenge

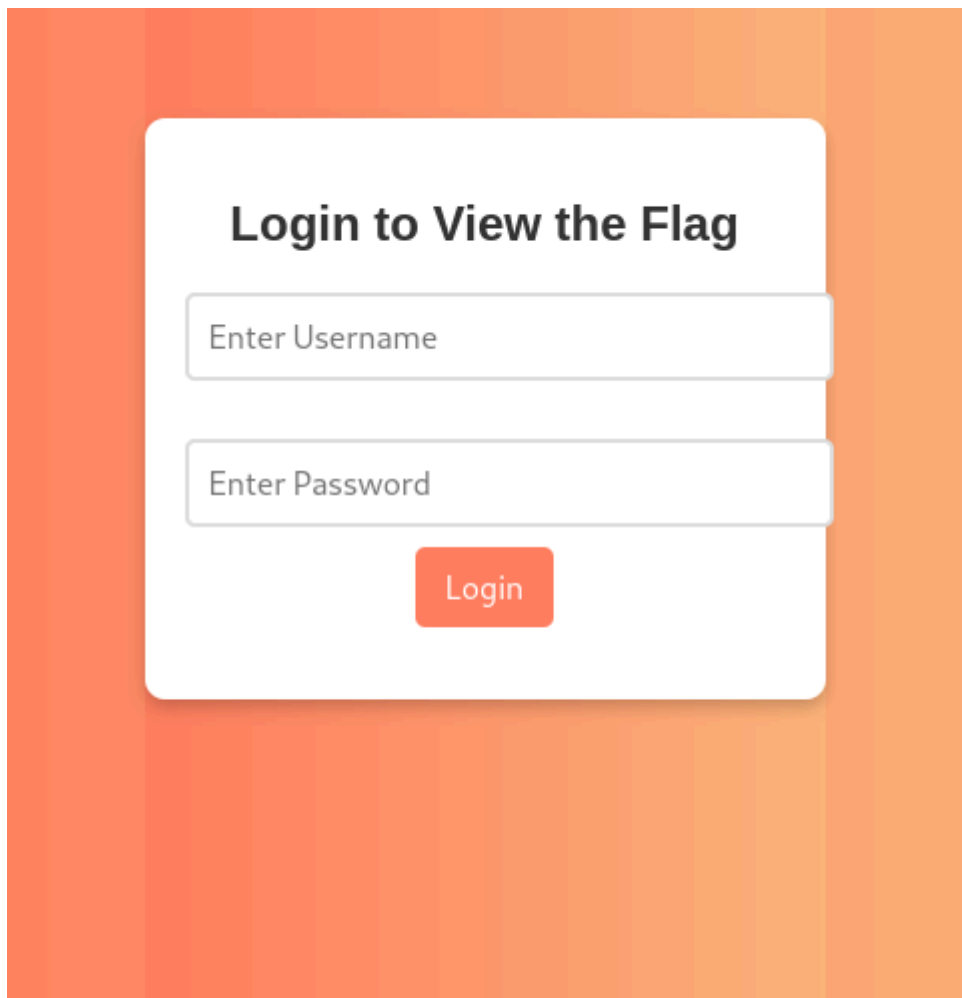
Description : To Login This Page And Get Flag

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.

```
Search 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.

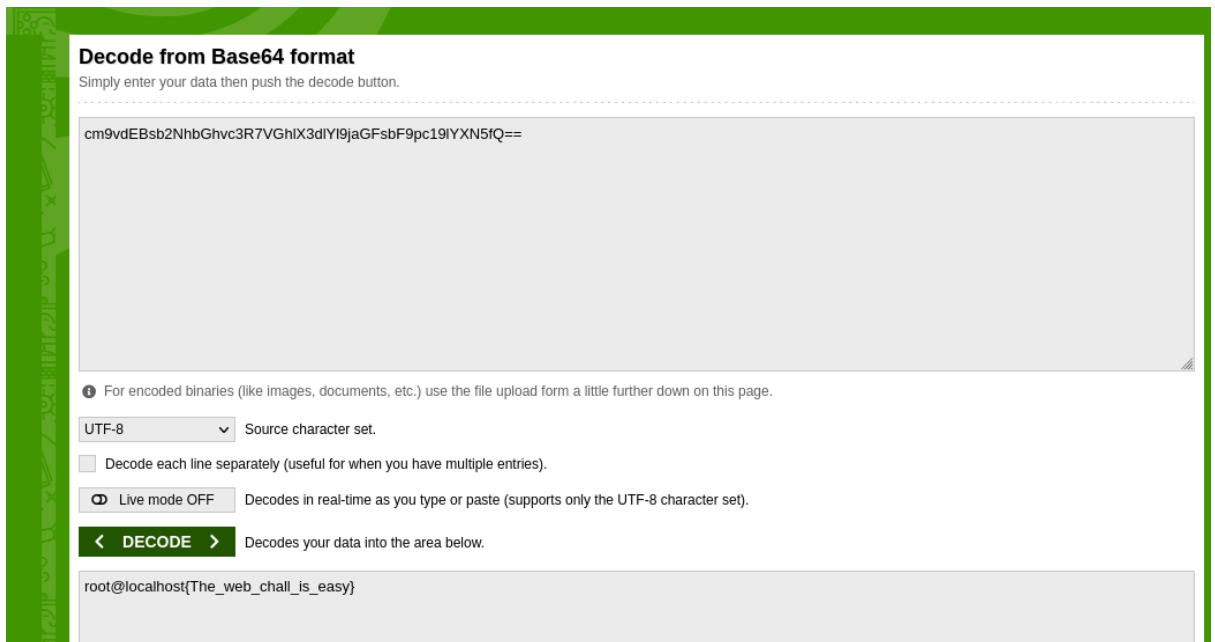
Finding the Flag

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

```
Sources Outline Search
Main Thread
web-chall-ten.vercel.app
JS script.js
1 const validUsername = 'root';
2 const validPassword = 'Helloworld';
3 const encodedFlag = 'cm9vdEBsb2NhbkGhvc3R7V6hLX3dLYl9jaGFsbF9pc19lYXN5fQ==';
4
5 function checkCredentials() {
6   const username = document.getElementById('username').value;
7   const password = document.getElementById('password').value;
8
9   if (username === validUsername && password === validPassword) {
10     const flag = atob(encodedFlag); // Decode Base64 encoded flag
11     document.getElementById('flag').innerText = flag;
12     document.getElementById('flag').style.display = 'block';
13   } else {
14     alert('Incorrect username or password!');
15   }
16 }
17
```

Decoding the Flag

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

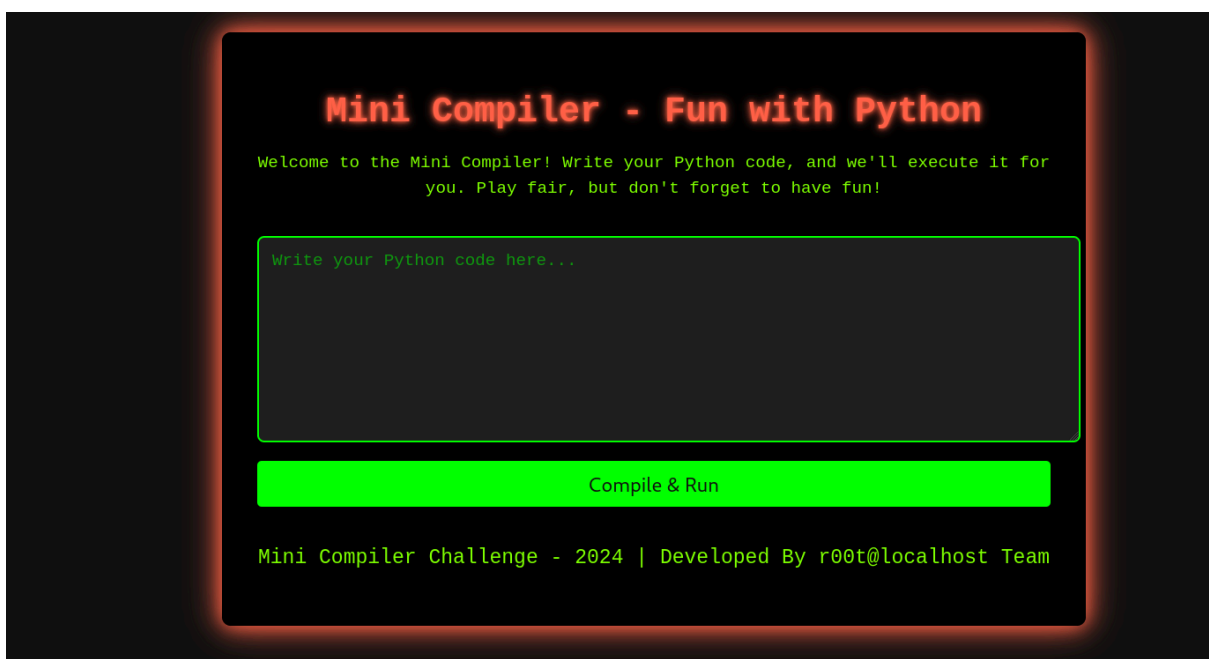


The screenshot shows a web interface for decoding Base64 data. At the top, it says "Decode from Base64 format" and "Simply enter your data then push the decode button." Below this is a large text input area containing the base64 string: `cm9vdEBsb2NhbGhvc3R7VGlIX3dlY9jaGFsbF9pc19lYXN5fQ==`. Under the input area, there are several options: a dropdown menu set to "UTF-8" with the label "Source character set.", a checkbox for "Decode each line separately (useful for when you have multiple entries).", and a checkbox for "Live mode OFF" with the description "Decodes in real-time as you type or paste (supports only the UTF-8 character set).". A green button labeled "< DECODE >" is positioned below these options, with the text "Decodes your data into the area below." to its right. At the bottom, a text output area displays the decoded result: `root@localhost{The_web_chall_is_easy}`.

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:



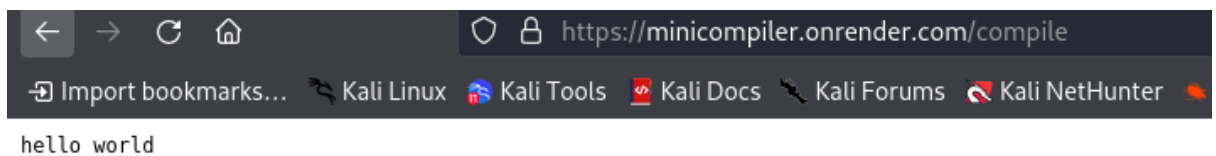
The screenshot shows a web interface titled "Mini Compiler - Fun with Python". Below the title, a welcome message reads: "Welcome to the Mini Compiler! Write your Python code, and we'll execute it for you. Play fair, but don't forget to have fun!". There is a large text input area with the placeholder text "Write your Python code here...". Below the input area is a green button labeled "Compile & Run". At the bottom of the interface, a footer message says: "Mini Compiler Challenge - 2024 | Developed By r00t@localhost Team".

Accessing the Website

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

Testing Code Execution

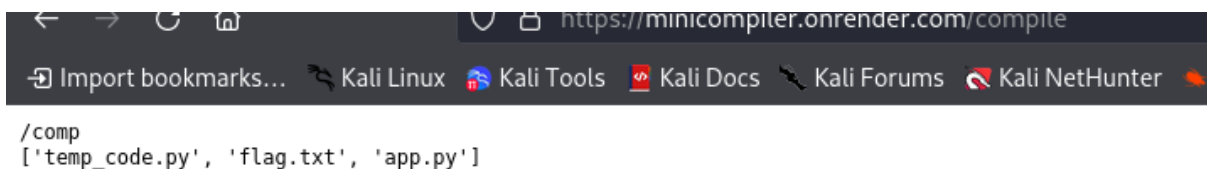
- Entered a simple Python command: `print('hello world')`.



- 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

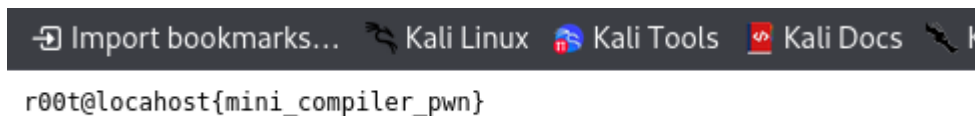
- Discovered a file named `flag.txt` in the current directory.



- 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.

iDoor

Customer ID: 20

Camera Status: inactive

Flag: root@localhost{this_is_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.



f5ca38f748a1d6eaf726b8a42fb575c3c71f1864a8143301782de13da2d9202b

This appears to be a hexadecimal string, likely representing a hash value or a unique identifier. It could be a hash of a password, a file, or some other piece of data.

Common Hash Algorithms:

- MD5: 32-character hexadecimal string
- SHA-1: 40-character hexadecimal string
- SHA-256: 64-character hexadecimal string
- SHA-512: 128-character hexadecimal string

Possible Uses:

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 for 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.

Encrypter Decrypter

SHA256 Hash

f5ca38f748a1d6eaf726b8a42fb575c3c71f1864a8143301782de13da2d9202b

Text

20

»

Elapsed Time: 0.349s Trial Count

Decryption Settings > Decrypt > Reset

-

Experimenting with Parameter Values

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

Encrypter Decrypter

Text

19

»

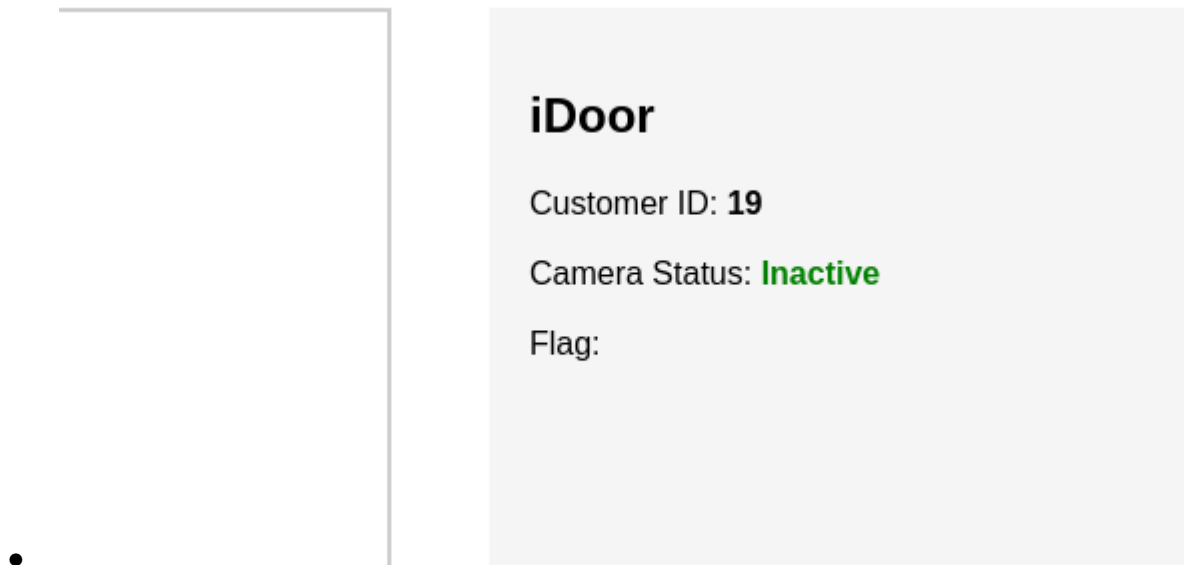
SHA256 Hash

9400f1b21cb527d7fa3d3eabba93557a18ebe7a2ca4e471cfe5e4c5b4ca7f767

Encrypt > Reset Copy

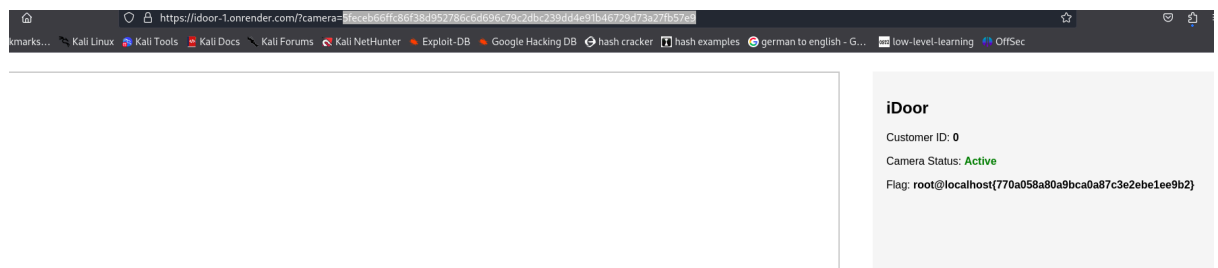
-

- 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.

"Find and exploit the XSS vulnerability "

what does it mean

Show drafts



Understanding "Find and Exploit the XSS Vulnerability"

Cross-Site Scripting (XSS) is a type of web security vulnerability that allows an attacker to inject malicious scripts into a webpage, often unnoticed by the user. When a vulnerable website reflects user-supplied input back to the user without proper sanitization or encoding, an attacker can exploit this to inject malicious JavaScript code.

The Exploitation Process:

1. Identifying Vulnerable Input Points:

- Look for places where user-supplied data is directly reflected in the output, such as comment fields, search boxes, or profile pages.

2. Crafting a Malicious Payload:

- Create a malicious script that will be executed when the user visits the vulnerable page. This script can perform various malicious actions, such as:
 - Stealing cookies and session tokens
 - Redirecting the user to malicious websites
 - Injecting malicious code into the page

- Defacing the website

- 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.

-

Your lower cased input is :

normal text with h tag

Your lower cased input is :

h5 text

-
- 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:
``

Give me something:

Submit

Retrieving the Flag

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

lower cased input is :

🌐 xss-j4in.onrender.com

root@localhost{Byp4ss_Sanitiz3r_123}

Cancel

OK

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**.

```
<div class="album py-5">
  <div class="container">
    <div class="row"> flex
      <div class="col-md-6 offset-md-3 text-center">
        <h1>Student care finder</h1>
        <br>
        <h3 id="result">Login</h3>
        <br>
        <form class="container" method="post">
          <label for="uname"> <input type="text" placeholder="Enter Username" name="user" required="">
          <br>
          <label style="padding-top: 10px" for="psw"> <input type="password" placeholder="Enter Password" name="pass" required="">
          <br>
          <br>
          <!--Username: root-->
          <button type="submit">Login</button>
        </form>
      </div>
    </div>
  </div>
</main>
</body>
</html>
```

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.

Learn about the upcoming changes to jwt.io and share your feedback →

JWT

Debugger Libraries Introduction Ask

Crafted by Auth0 by Okta

Algorithm: HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoicm9vdCJ9.0E8L070IiY4ZADEiIs9fGzP-Tz1_F3yqu0RcYzME9k
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "user": "root"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + ".",
  base64UrlEncode(payload),
  101
)
☐ secret base64 encoded
```

Signature Verified

SHARE JWT

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.

https://web2-k7a3.onrender.com

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB hash cracker hash examples german to english - G... low-level-learning OffSec

Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application Cookie-Editor

Cache Storage

Cookies

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoicm9vdCJ9.0E8L070IiY4ZADEiIs9fGzP-Tz1_F3yqu0RcYzME9k	web2-k7a3...	/	Session	106	false	false	None	Mon, 09 Dec 2024 07:09:48 ...

Indexed DB

Local Storage

Session Storage

Successfully Gaining Access

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

https://web2-k7a3.onrender.com/dashboard

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB hash cracker hash examples german to english - G... low-level-learning OffSec

SYSTEM DASHBOARD

YOU ARE LOGGED IN AS USER **root**.

System Status:

Welcome, root@localhost! Here is your flag:
root@localhost{P@ssw0rDS_r_0pti0n4l}