Web Application Security

Assignment 1:

Objective:

To Perform a SQL injection attack on a vulnerable web application (such as DVWA or OWASP Juice Shop) and extract information from the database.

Performing a SQL injection attack involves exploiting vulnerabilities in a web application's input validation to execute malicious SQL queries and extract data from the database. I performed in DVWA (Damn Vulnerable Web Application) set up in a controlled lab environment.

- SQL Injection:
    - Classic/Error-Based: uses abnormal inputs to provoke database error messages or altered queries.
    - Blind SQLi: relies on boolean or time-based responses when error messages aren't exposed.
    - Union-Based: leverages UNION queries to combine results from different Select statements.

SQL Injection Attack on DVWA (Damn Vulnerable Web Application)

1. **Setup**

- DVWA (Damn Vulnerable Web Application) is a deliberately insecure web application intended for security training.
- Running in a local, isolated environment at the low security level provides a safe setting to learn about common web vulnerabilities, including input handling and basic exploitation concepts, without impacting external systems.

Environment and prerequisites

- A Linux distribution (e.g., Ubuntu) or Windows with a local stack (e.g., XAMPP) or a containerized environment (Docker) to host a web server and a database.
- A local web server (Apache or equivalent) and a database server (MySQL/MariaDB) configured to serve DVWA.
- DVWA files downloaded from its repository and placed in the webroot (e.g., /var/www/html/dvwa or equivalent in XAMPP's htdocs).

Typical setup steps (high level, non-destructive)

- Install and start a web server and database service.
- Deploy DVWA into the webroot and configure the database connection with a dedicated DVWA database user.
- Adjust PHP and server configuration as needed to enable DVWA (e.g., database access, required PHP modules).
- Open the DVWA setup page in a browser and initialize the database, then log in with the provided credentials (often admin/password or configured during setup).
- Set the DVWA security level to low for an educational baseline, taking care to note the boundaries of the lab (no external access).

2. Identify Injection Point

- Navigate to the SQL Injection page in DVWA.
- Typically, there is a form or URL parameter where user input is sent directly to database queries.

3. Basic Injection Attempts

- Common payloads to test:
    - 1' OR '1'='1 (to bypass login or retrieve all data)

4. Extract Data

- Input the payload in the vulnerable input field and submit.
- If vulnerable, the database will return more data than intended, such as user credentials.

5. Example Payload:

text

1' OR '1'='1

This crafts a SQL query like:

sql

SELECT * **FROM** users **WHERE** id = '1' OR '1'='1';

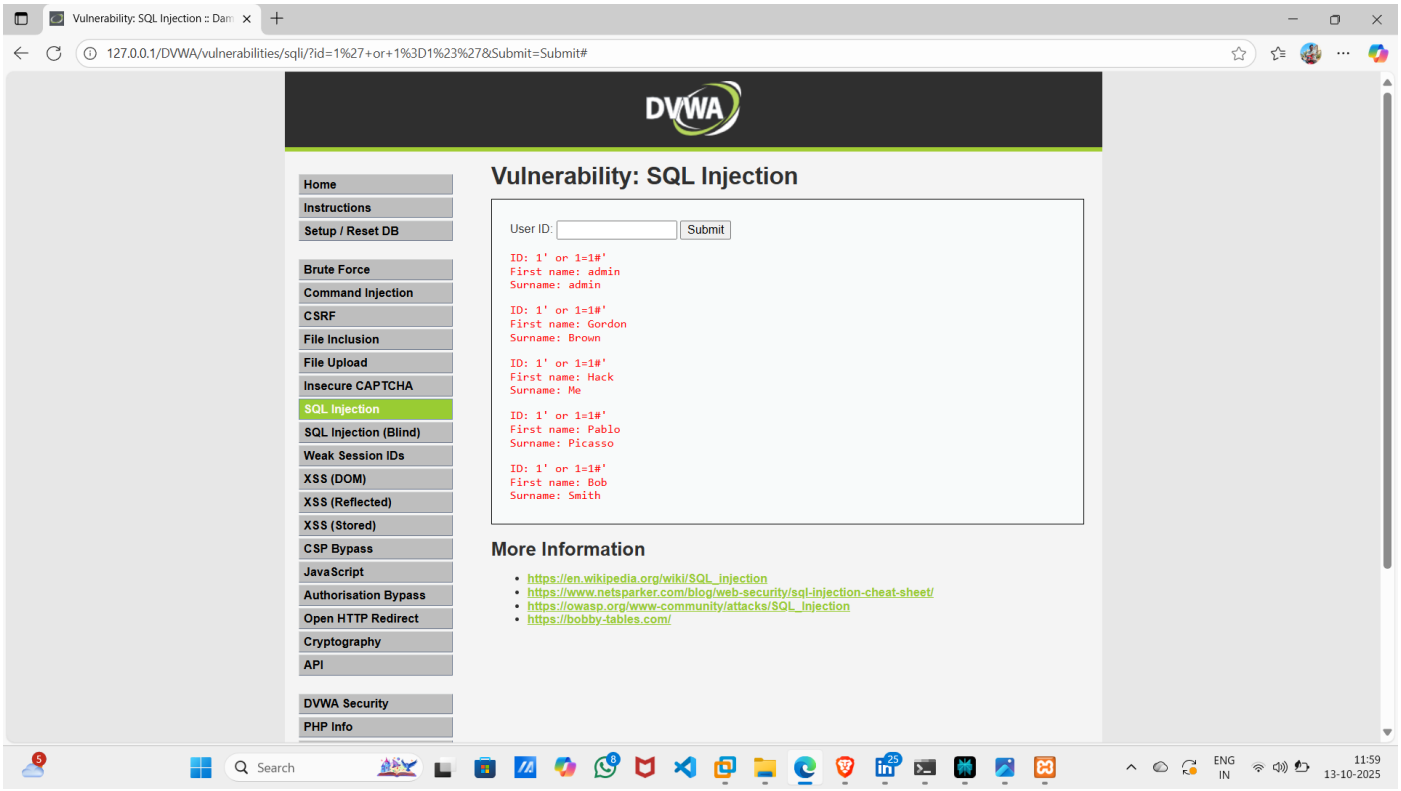
which returns all users.

ID: 1' or 1=1#'
First name: Gordon
Surname: Brown

ID: 1' or 1=1#'
First name: Hack
Surname: Me

ID: 1' or 1=1#'
First name: Pablo
Surname: Picasso

ID: 1' or 1=1#'
First name: Bob
Surname: Smith



This POC

Conclusion:

Here I get all the user data.

I used payload 1’ or 1=1#’

Assignment 2:

Objective:

Perform a Cross-Site Scripting (XSS) attack on a vulnerable web application by injecting a JavaScript alert (<script>alert(“XSS”);</script>) into a form field.

Introduction:

Cross-Site Scripting (XSS) is a web vulnerability that allows an attacker to inject malicious client-side code (usually JavaScript) into pages viewed by other users. This can lead to cookie theft, session hijacking, defacement, or redirection to malicious content, depending on how the vulnerable site processes and renders user input.

Key XSS types

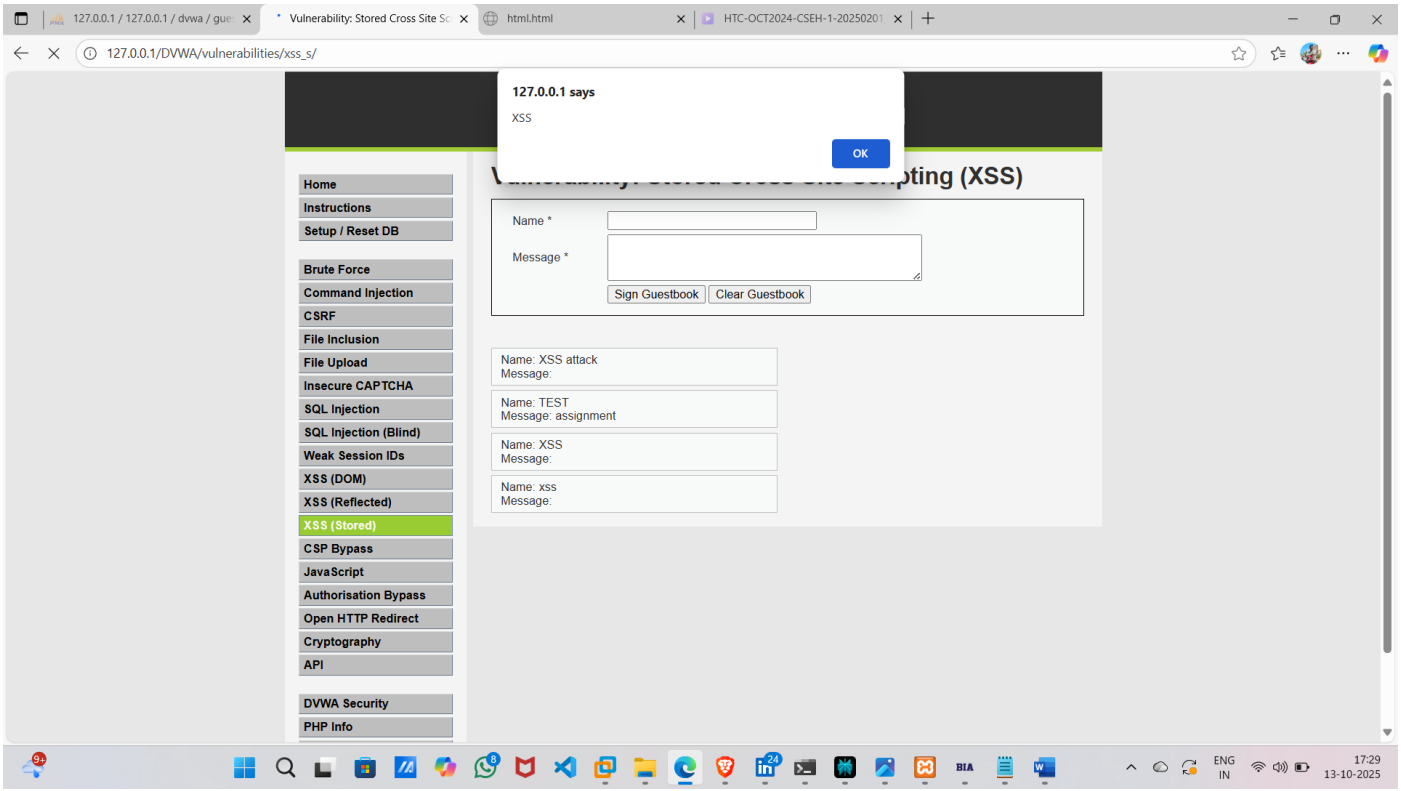
- **Reflected (non-persistent) XSS:** The attack payload is embedded in a request (e.g., in a URL or form submission) and reflected by the server in the immediate response. The user must interact with a crafted link or input to trigger it.
- **Stored (persistent) XSS:** The payload is stored on the server (e.g., in a database, comment field, or user profile) and served to any user who views the affected page.
- **DOM-based XSS:** The payload is executed entirely in the browser as a result of client-side JavaScript manipulating the DOM, without server-side reflection or storage of the payload.

Set-up the Environment

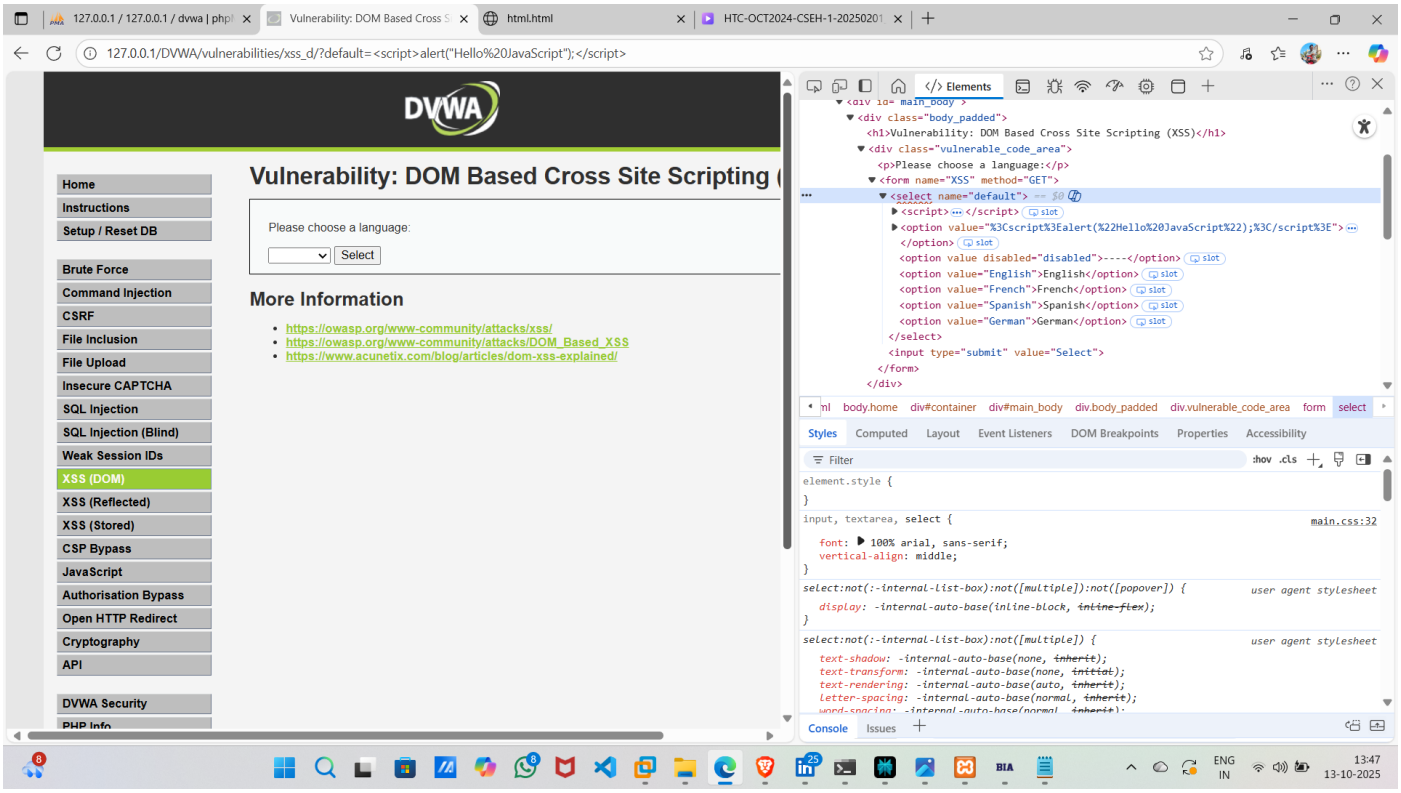
As shown in above assignment 1:

I performed the Javascript (<script>alert(“XSS”);</script>).

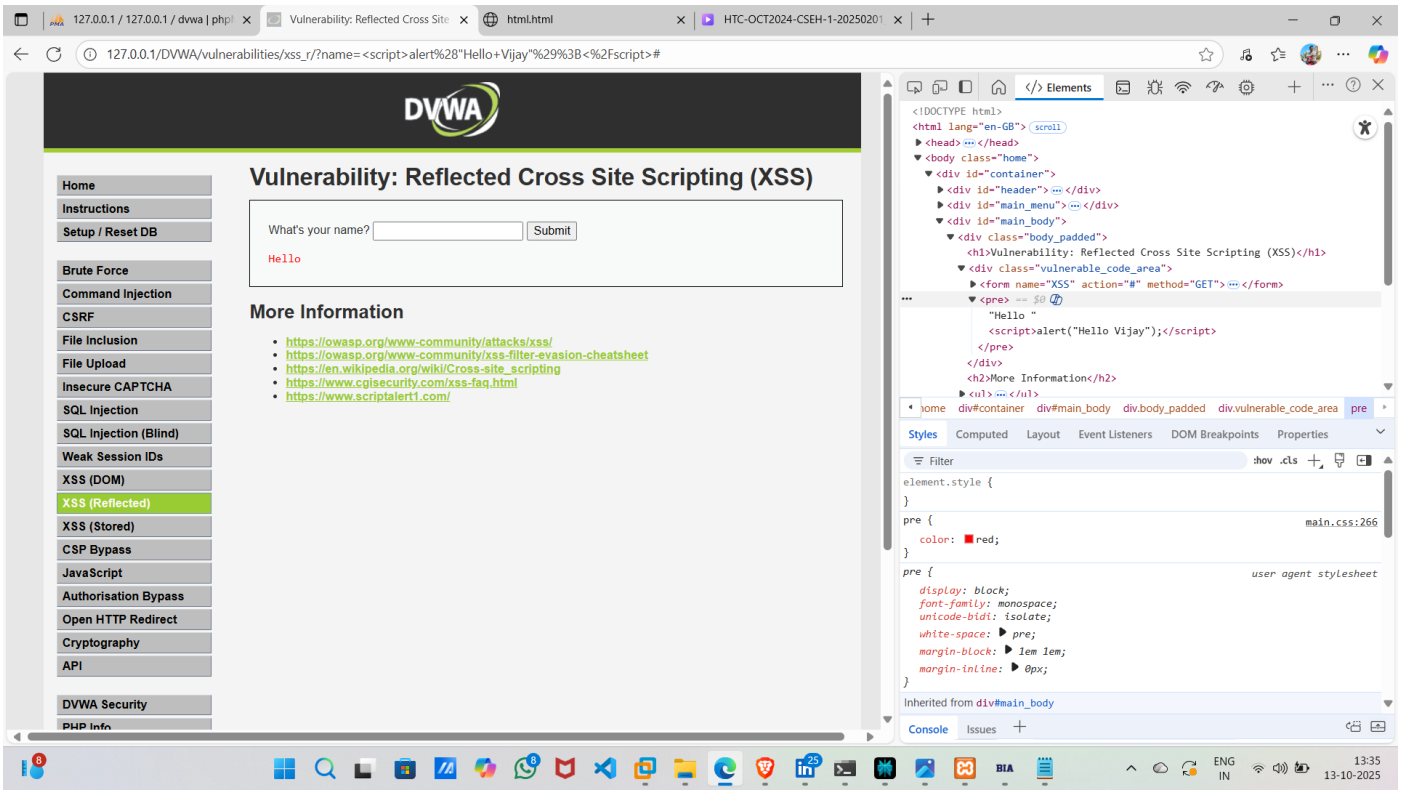
Evidence and Artifacts



Stored XSS



DOM XSS



Reflected XSS

Conclusion:

I performed the same script in all three different types of XSS. Above are the Proof of Concept showing how they react when I injected the code into in different types.

Every time I injected the code it shows pop-up as what I placed in between the alert code of script.

And I inspect the page every-time.