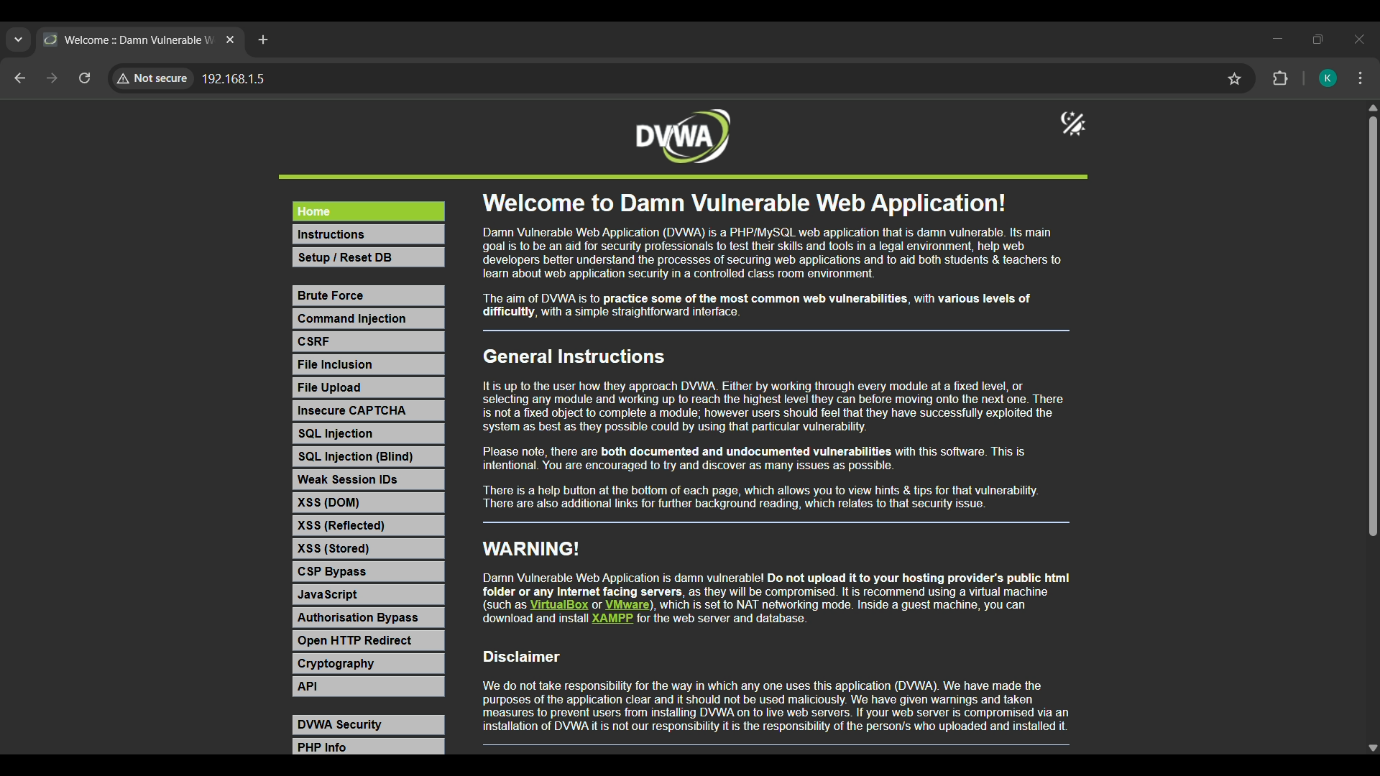
**Cybersecurity Internship Report: Web Application Security Testing – Task 1**

**Application Tested:** DVWA (Damn Vulnerable Web Application)  
**Tools Used:** OWASP ZAP, Burp Suite, SQLMap.  
**Tested By:** Jeet Nishad  
**Email:** jeetnishad12@gmail.com

**Introduction to DVWA:**

DVWA (Damn Vulnerable Web Application) is a PHP/MySQL web application that is intentionally designed to be insecure. Its primary purpose is to provide a legal environment for security professionals, students, and researchers to practice and improve their penetration testing skills. It includes multiple vulnerability levels for different attack types such as SQL Injection, Cross-Site Scripting (XSS), Brute Force, and more.

In this project, I performed manual and automated testing on DVWA using various tools to identify and exploit common web application vulnerabilities. This report details the vulnerabilities found, how they were exploited, and how they can be mitigated.



**1. Introduction**

This report summarizes the findings of a web application security testing task performed on **DVWA (Damn Vulnerable Web Application)**. The objective of this task was to identify and understand common web application vulnerabilities using real-world techniques and automated tools. The vulnerabilities tested in this assessment include **SQL Injection**, **Cross-Site Scripting (XSS)**, and **Authentication Flaws**.

**2. Tools Used**

The following tools were used in conducting the security assessment:

* **Burp Suite**: A web vulnerability scanner and proxy tool for identifying security flaws in web applications.
* **SQLMap**: An automated tool for detecting and exploiting SQL injection vulnerabilities.
* **OWASP ZAP**: An open-source tool for finding security vulnerabilities in web applications.
* **DVWA (Damn Vulnerable Web Application)**: The target web application used for testing.

**3. Methodology**

The security assessment followed a structured methodology to test for common web vulnerabilities:

1. **Reconnaissance**:  
   Initial scanning of the application to identify attack surfaces and potential weaknesses.
2. **Vulnerability Scanning**:  
   Automated tools like **Burp Suite** and **OWASP ZAP** were used to scan the application for known vulnerabilities.
3. **Exploitation**:  
   Exploiting identified vulnerabilities using **SQLMap** for SQL injection and manual testing for XSS vulnerabilities.
4. **Reporting**:  
   Documenting the vulnerabilities found, the exploitation methods used, and recommended mitigation strategies.

**4. Vulnerabilities Identified**

**4.1 SQL Injection (SQLi)**

**Description**:  
SQL Injection is a common web application vulnerability where malicious SQL code is injected into a query, potentially allowing unauthorized access to the database.

**Test Method**:

* Manual exploitation and use of **SQLMap** to test for SQL injection points.
* Success was achieved by injecting SQL queries in the input fields that were not sanitized properly.

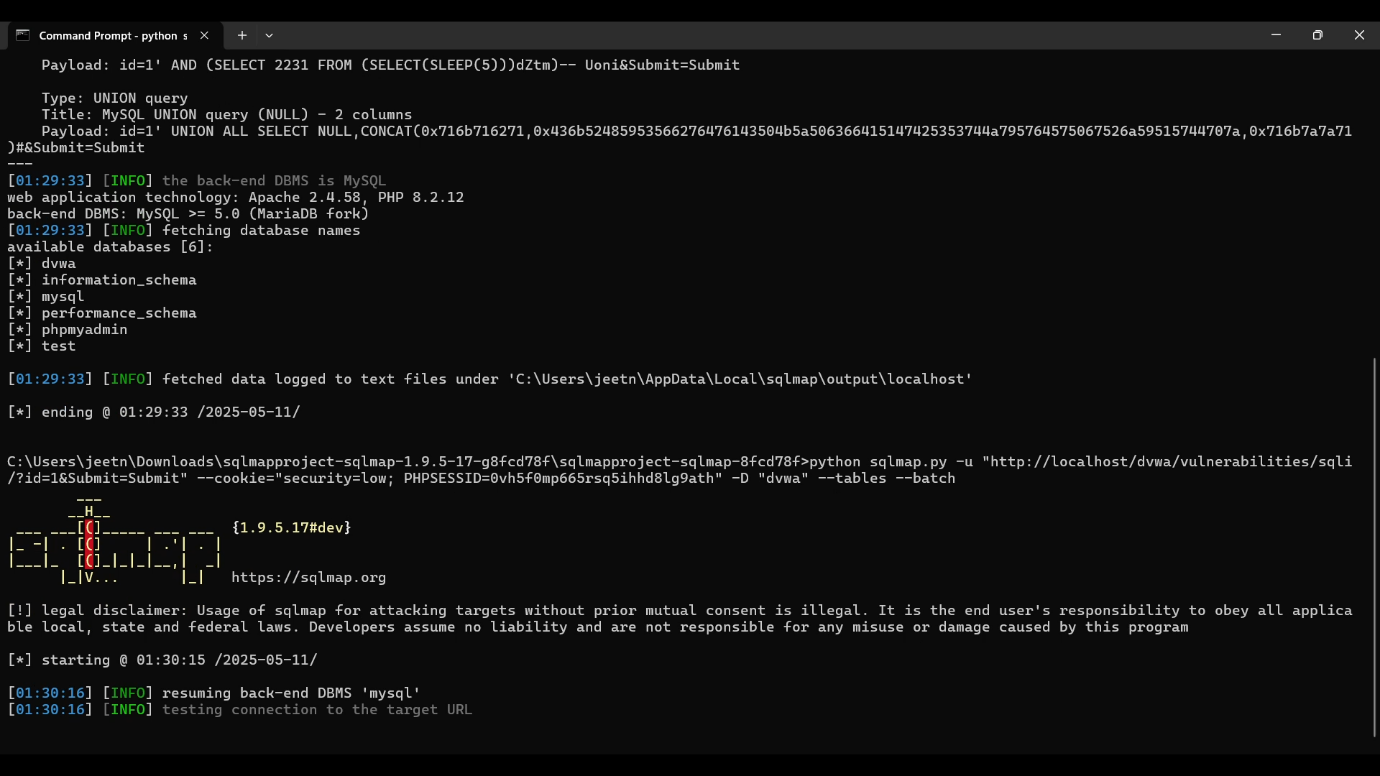
**Example of Attack**:

* Inputting ' OR '1'='1 in the login form resulted in unauthorized access to the admin panel.

**Mitigation Strategy**:

* Use parameterized queries and **prepared statements** to prevent SQL injection.
* Validate and sanitize user input rigorously.

**Screenshot**:  
Here’s an image demonstrating the **successful SQL injection**:



**4.2 Cross-Site Scripting (XSS)**

**Description**:  
Cross-Site Scripting (XSS) vulnerabilities allow attackers to inject malicious scripts into web pages, which are executed in the context of the user's browser.

**Test Method**:

* Identifying potential XSS vulnerabilities by injecting malicious scripts into form fields.
* **Reflected XSS** was tested by injecting a script into the input field to steal the session ID.

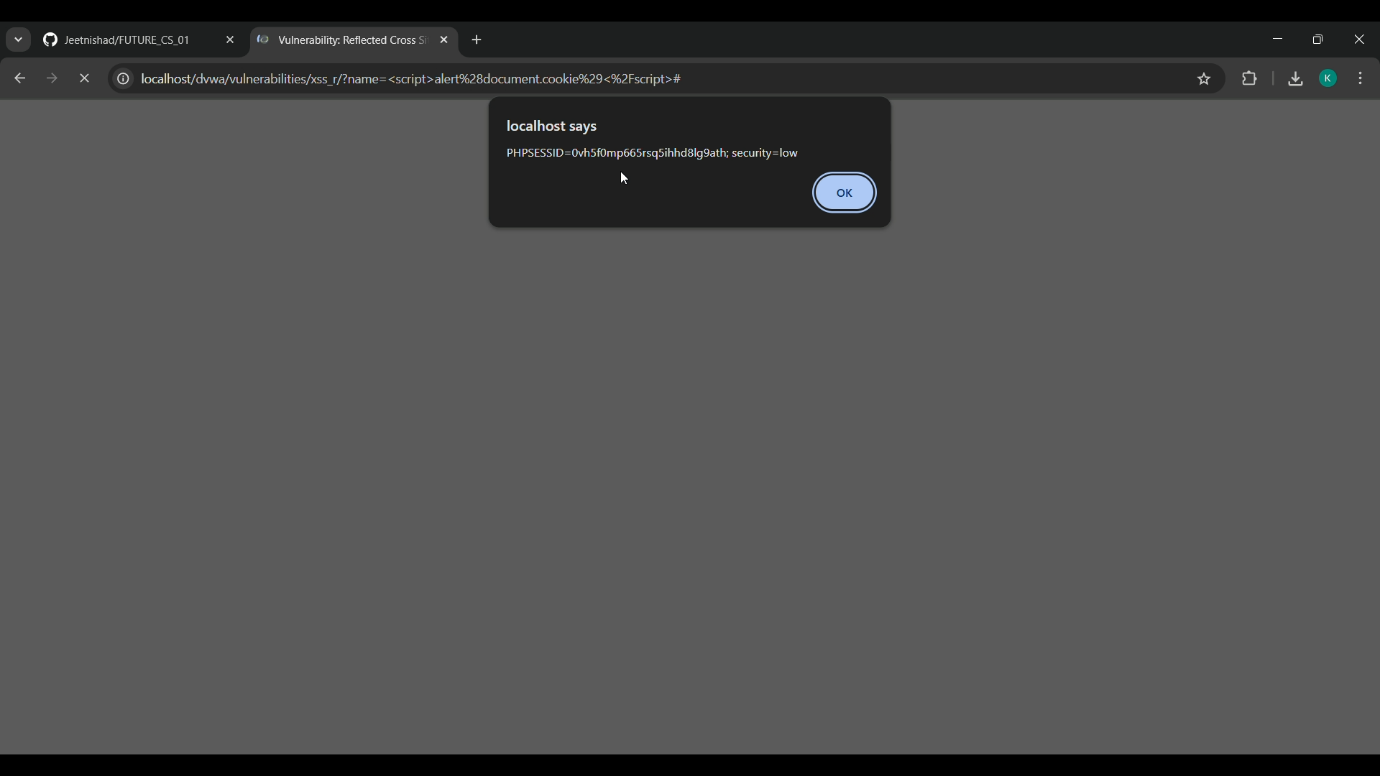
**Example of Attack**:

* Injecting <script>alert('XSS')</script> into the search input field displayed a pop-up in the browser.

**Mitigation Strategy**:

* Sanitize all user input to remove harmful scripts.
* Use content security policies (CSP) and HttpOnly cookies to protect against script execution.

**Screenshot**:  
Here’s an image demonstrating the **XSS attack** in action:



**4.3 Authentication Flaws (Brute-Force Attack Using Burp Suite)**

Description:

Authentication mechanisms are the front line of defense for any web application. Weaknesses in authentication systems can allow attackers to gain unauthorized access through brute-force attacks, default credentials, or insecure login flows.

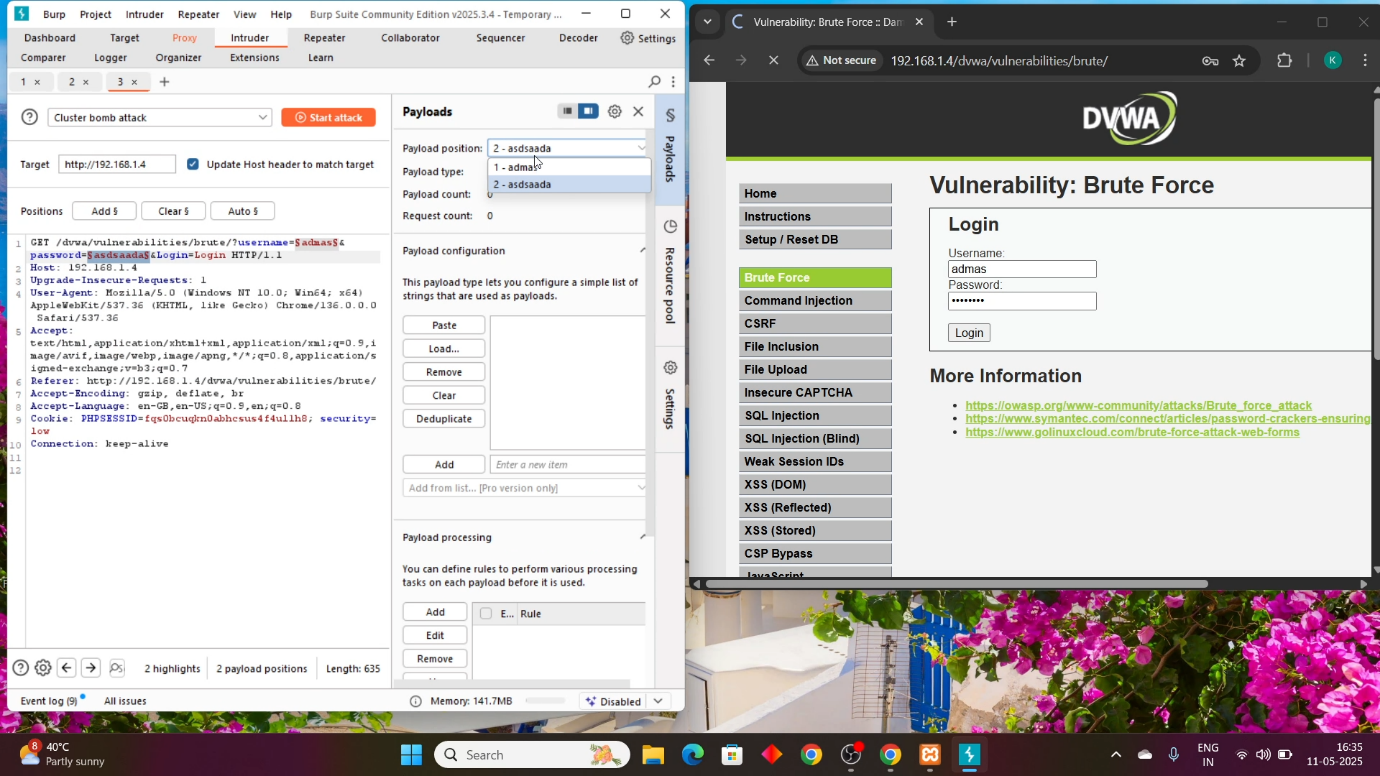
Test Case:

To assess authentication flaws in DVWA, I targeted the login functionality using a brute-force attack. The steps were as follows:

1. Intercepted the Login Request
   * Used Burp Suite to intercept a valid login attempt.
   * Captured the POST request structure with parameters username and password.
2. Configured Burp Intruder
   * Set the attack type to Sniper or Cluster Bomb depending on whether one or both fields were targeted.
   * Loaded a list of common usernames and passwords (e.g., admin, password, 123456, etc.).
3. Launched the Brute-Force Attack
   * Observed changes in server response (status codes or content length).
   * Identified a successful login when the server responded with a different pattern or redirected to the dashboard.

Result:

* DVWA allowed unlimited login attempts without CAPTCHA or account lockout.
* Successfully brute-forced valid credentials.
* Session token was issued without any security flags (Secure, HttpOnly, SameSite).

Insert Screenshot Here:  


Risk Level: High

Authentication flaws such as unrestricted brute-force can allow attackers to compromise user accounts, especially when users use weak or reused passwords.

**5. Summary of Findings**

The security assessment of the **DVWA** revealed the following vulnerabilities:

* **SQL Injection** in unfiltered input fields.
* **Cross-Site Scripting (XSS)** vulnerability in user input handling.
* **Authentication flaws** related to session management.

**6. Mitigation Recommendations**

* **SQL Injection**:
  + Always use parameterized queries and prepared statements.
  + Apply proper input validation to prevent malicious input.
* **XSS**:
  + Sanitize user inputs by escaping special characters.
  + Use security mechanisms like Content Security Policy (CSP) and HttpOnly cookies.
* **Authentication Flaws**:
  + Implement strong password policies and session management.
  + Use multi-factor authentication (MFA) wherever possible.

**7. Conclusion**

This task helped me to understand common web application vulnerabilities and how they can be exploited in real-world scenarios. Using automated tools and manual techniques, I was able to successfully identify vulnerabilities in the **DVWA** application. The task also provided valuable experience in performing web application security assessments and recommending security improvement.

**8.Contact Information**

**Author**: Jeet Nishad  
**Email**: jeetnishad12@gmail.com  
**Date**: April 2025