TASK 1
WEB APPLICATION SECURITY TESTING

INTERNSHIP BATCH: Future Interns
REPORT BY: Singari Keerthi

TABLE OF CONTENTS

Summary

Web application security is the practice of defending websites and online applications from cyberattacks by protecting the application's code, data, and infrastructure from vulnerabilities and unauthorized access. It involves implementing a range of strategies, technologies, and best practices throughout the software development lifecycle, including input validation, strong authentication, encryption, regular security testing, and patch management, to ensure data confidentiality, integrity, and availability.

This exercise is a controlled, instructional security test of DVWA (local lab). The main goal was to detect and show typical web weaknesses (SQL Injection and Cross-Site Scripting), gather evidence, and suggest pragmatic mitigations.
The test verified:

1. SQL Injection in GET /DVWA/vulnerabilities/sqli/
2. Reflected & Stored XSS in DVWA

All testing was conducted against the deliberately vulnerable DVWA installation locally installed on Kali Linux.

Environment Setup

1. **Operating System:** Kali linux.

2. **Install DVWA:**

   ```
   cd /var/www/html
   sudo git clone https://github.com/digininja/DVWA.git DVWA
   sudo chown -R www-data:www-data /var/www/html/DVWA
   sudo chmod -R 755 /var/www/html/DVWA
   ```

3. **Configure DB user:**

   ```
   sudo mysql -u root
   CREATE DATABASE dvwa;
   CREATE USER 'dvwa'@'localhost' IDENTIFIED BY 'dvwa';
   GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwa'@'localhost';
   FLUSH PRIVILEGES;
   EXIT;
   cd /var/www/html/DVWA/config
   sudo cp config.inc.php.dist config.inc.php
   ```

4. **Start Apache & MariaDB:**
   ```
   sudo service apache2 start
   sudo service mysql start
   ```

5. **Open DVWA in browser:**

   http://localhost/DVWA/login.php
   Username: admin
   Password: password

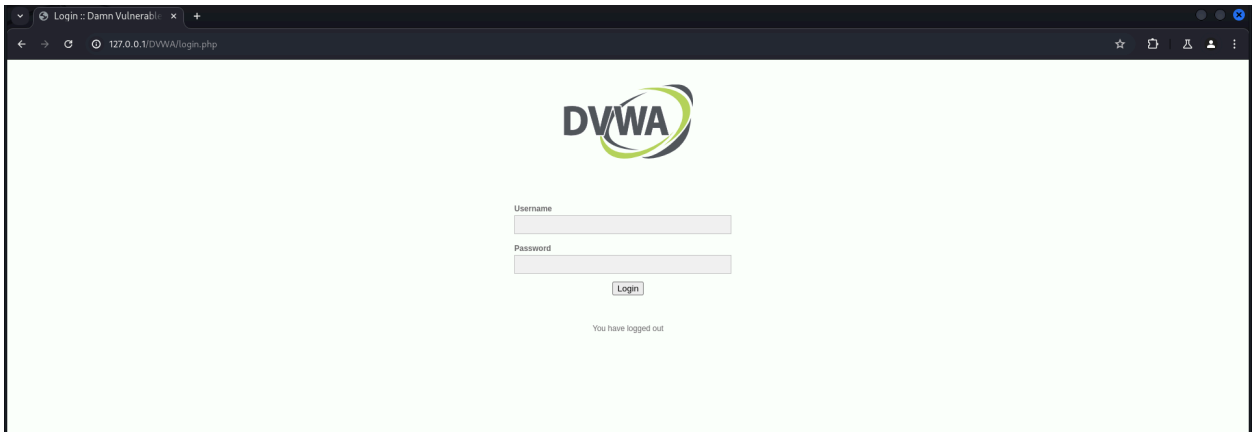

Figure 1: DVWA login page

Test methodology

1. **Recon & scope** — enumerate functionality and inputs: login forms, ID parameters, search, comment areas.

2. **Proxy capture** — configure Burp Suite as browser proxy and intercept requests.

3. **Manual testing** — inject common payloads in inputs / URL params and observe responses.

4. **Document** — capture screenshots and save request files from Burp.

Findings

1. IDOR (Insecure Direct Object Reference):

- Login to DVWA
- Tamper with the ID parameter
    - Change `id=1` → `id=4` in the request.
    - Forward request in Burp or directly in browser.

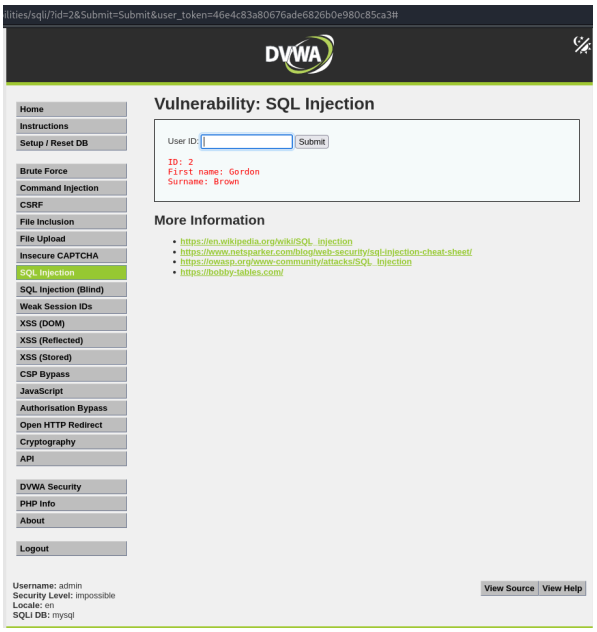

Figure 2: Credential Disclosure

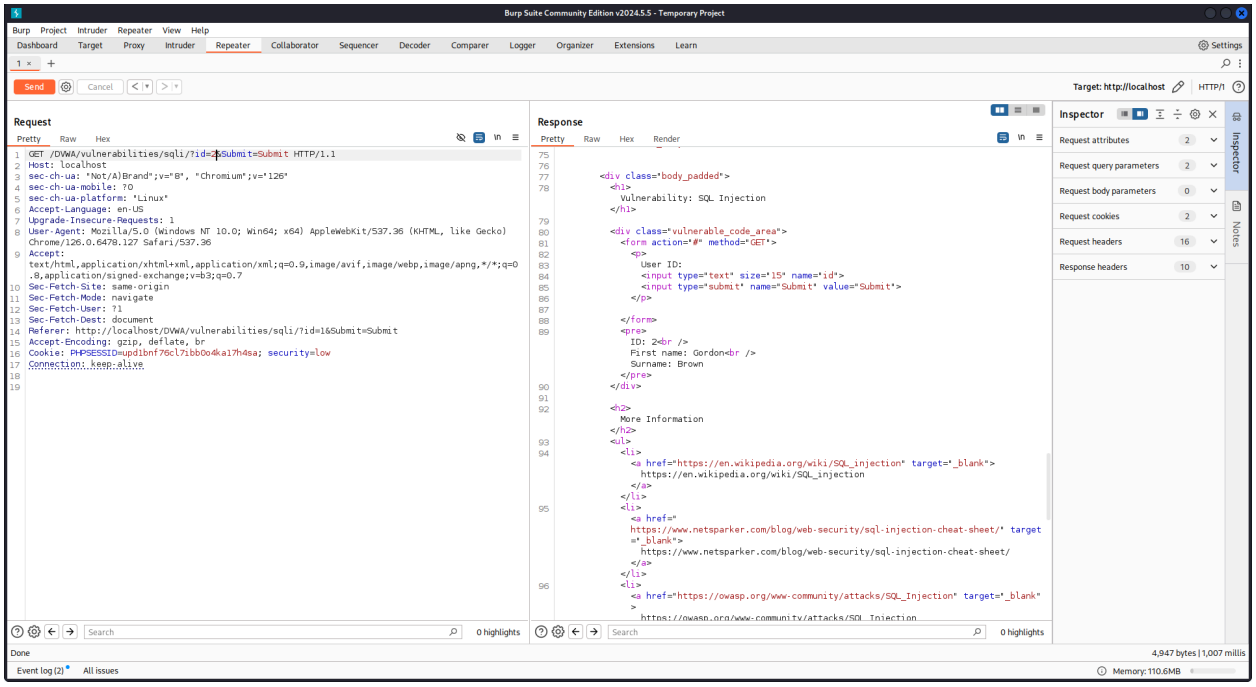

Figure 3: Captured in Burpsuite

Result:

Without any extra authentication, the application displays another user's details . This confirms that **access control is missing**, leading to IDOR.

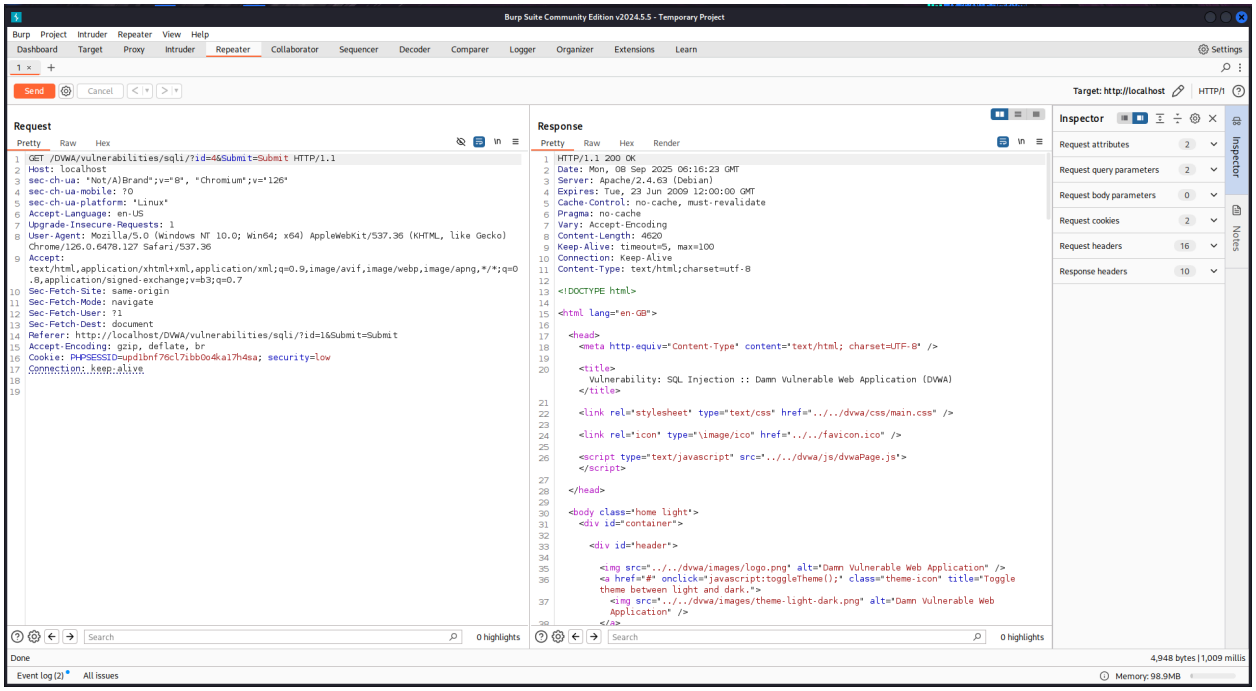

Figure 4: Credentials Disclosed in Burpsuite of Different Id.

2. SQL Injection:

- Open SQLi page: `http://127.0.0.1/DVWA/vulnerabilities/sqli/`
- Enter 1 in the **User ID** box and click Submit. Capture the request in Burp (Proxy → HTTP history).
- Send the captured request to **Repeater** (Right-click → Send to Repeater).
- Modify the parameter to cause an SQL error:
    - Change: `id=1` → `id=1 OR 1=1` (`id=1%200R%201=1`)
    - Send in the Repeater.

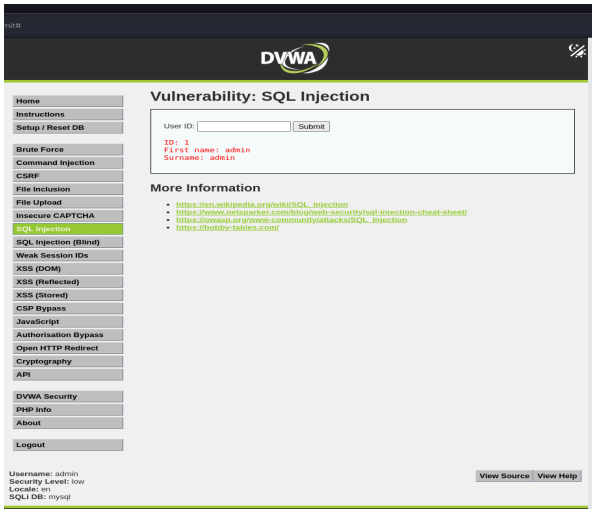

Figure 5: User Id Credentials

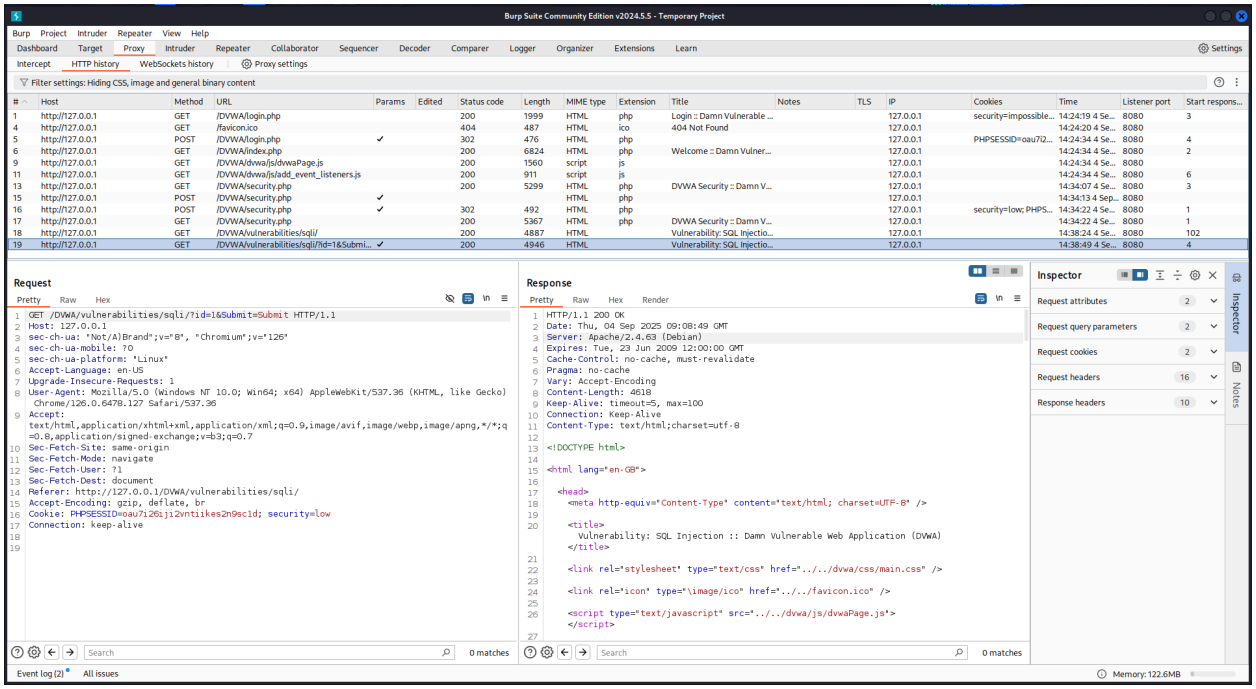

Figure 6: Captured in Burpsuite

Result:

<pre>ID: 1 OR 1=1
First name: admin
Surname: admin</pre>

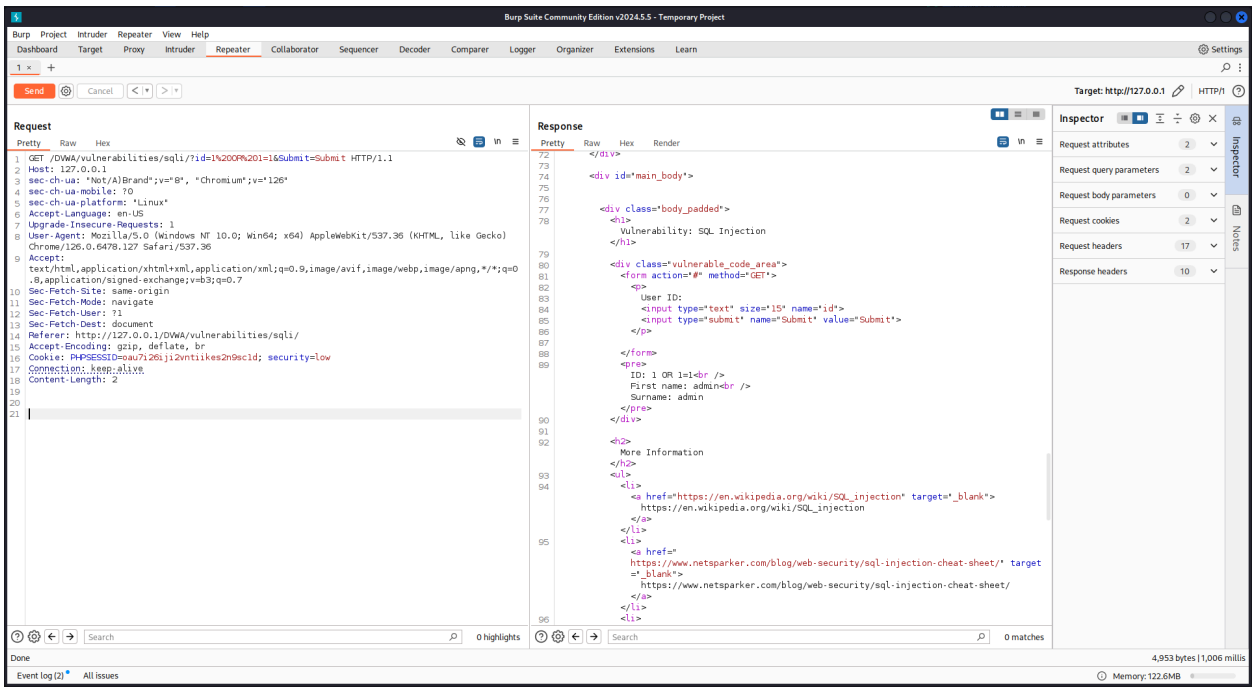

Figure 7: Payload Entered in ID

3. **Cross-Site Scripting (XSS):**

   (Reflected XSS)

- Go to `vulnerabilities/xss_r`.
- Enter payload:
  <script>alert('XSS')</script>
- Submit.

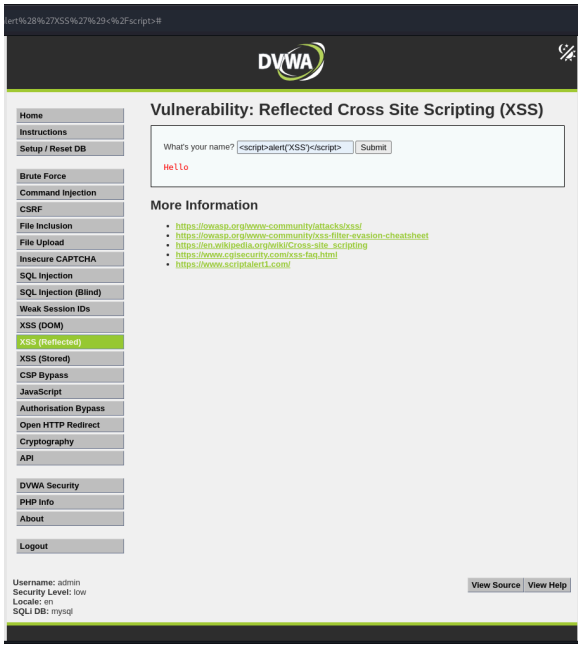

Figure 8: Payload Entered

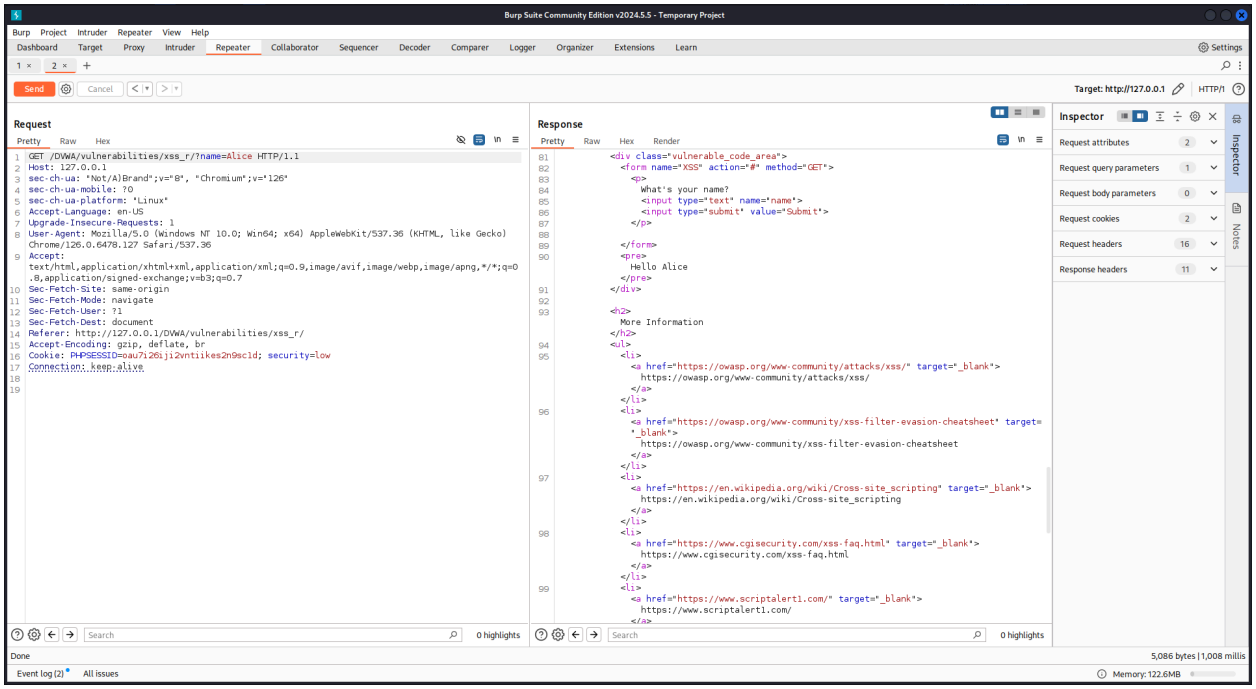

Figure 9: Captured in Bursuite

Result:

Alert in the browser when the payload is reflected.

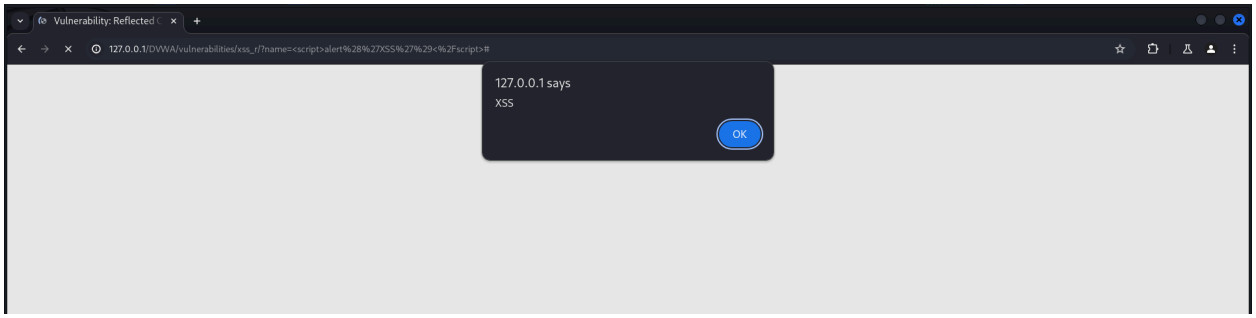

Figure 10: Alert of Reflected XSS

(Stored XSS)

- Go to a comment or feedback input (`vulnerabilities/xss_s`).
- Post the comment:
  <script>alert('XSS stored')</script>
- Reload the page.

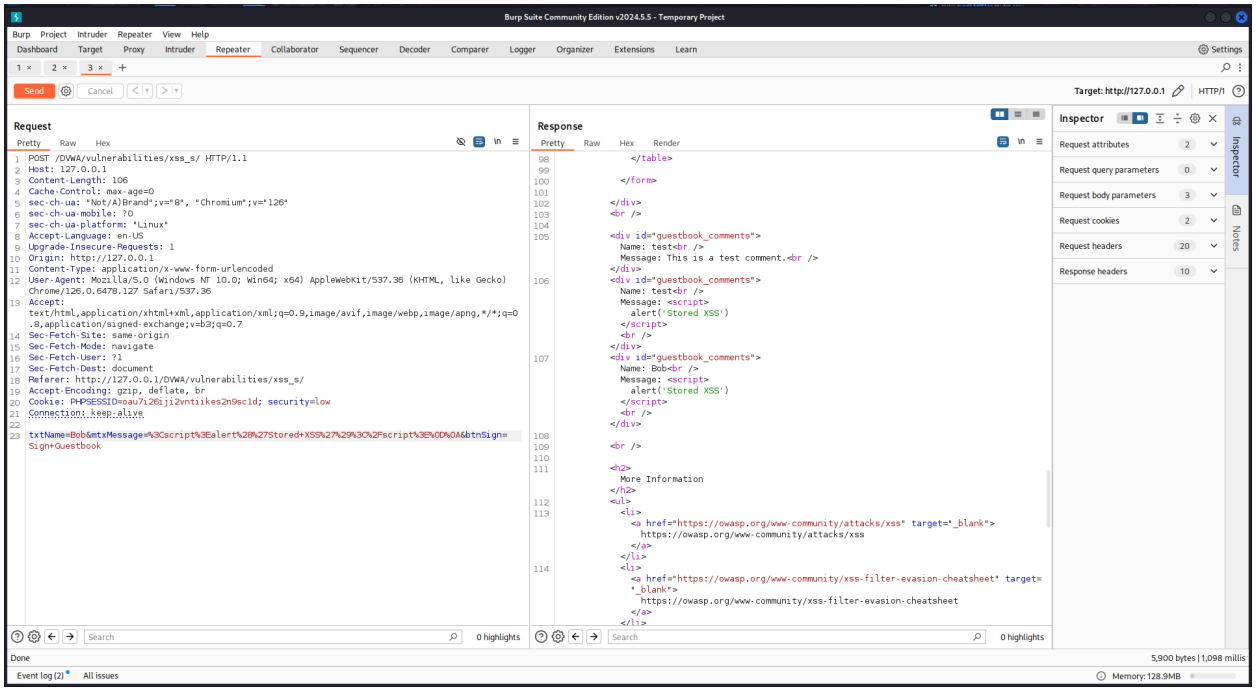

Figure 11: Captured in Burpsuite

Result:

The alert executes.

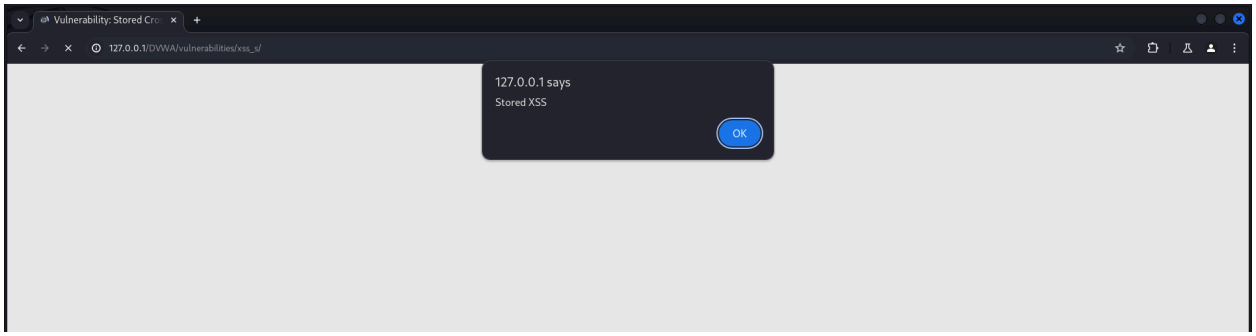

Figure 12: Alert of Stored XSS

Findings table

Vulnerability	Location	Severity	Recommended Remediation
Insecure Direct Object Reference (IDOR)	`/DVWA/vulnerabilities/idor/?id=`	High	Enforce server-side access control checks, use indirect references (UUIDs), implement RBAC
SQL Injection (SQLi)	`/DVWA/vulnerabilities/sqli/?id=`	High	Use parameterized statements, strict input validation, least-privileged DB accounts
Reflected XSS	`/DVWA/vulnerabilities/xss_r/`	Medium	Output encoding (HTML entities), input sanitization, implement CSP
Stored XSS	`/DVWA/vulnerabilities/xss_s/`	Medium	Output encoding, restrict stored input, sanitize before rendering

Mitigation

- Always use parameterized queries.
- Validate inputs especially for numeric IDs.
- Employ a Web Application Firewall (WAF) as an additional layer.

- Use secure templating libraries that auto-escape.

Conclusion

The security test of DVWA (Damn Vulnerable Web Application) was able to clearly illustrate how typical web application flaws can be discovered and used in a contained laboratory setting. The testing included SQL Injection, Cross-Site Scripting (XSS), and Insecure Direct Object Reference (IDOR), which all directly translate to key risks in the OWASP Top 10.

SQL Injection permitted database manipulation by adding specially crafted payloads (', OR 1=1). This demonstrated the lack of input validation and parameterized queries. Cross-Site Scripting (XSS) was replicated in both reflected and stored contexts, demonstrating that user inputs not sanitized could run arbitrary scripts in victims' browsers. Insecure Direct Object Reference (IDOR) illustrated poor access control, with unauthorized access to another user's details by altering the id parameter. These issues demonstrate the actual-world effect of flawed coding methods, inadequate access control, and absence of safe development procedures. While DVWA is specifically designed to be insecure, the vulnerabilities shown reflect those commonly targeted in production code.

The most important lessons from this evaluation are:The importance of secure coding techniques like parameterized queries, adequate input validation, and output encoding.The importance of strong access control measures to limit horizontal and vertical privilege escalation.

The benefits of continuous security testing with a combination of manual methods (Burp Suite, payload manipulation) and automated tools (OWASP ZAP).Through the adoption of the suggested mitigation measures (prepared statements, CSP, RBAC, secure session management), organizations can effectively minimize the attack surface of their web applications.This project reemphasized hands-on skills in web penetration testing and highlighted the significance of using a "security by design" methodology in every phase of web application development.

Reference

1. https://owasp.org/www-community/attacks/SQL_Injection
2. https://portswigger.net/web-security/cross-site-scripting
3. https://owasp.org/Top10/
4. https://owasp.org/www-community/Insecure_Direct_Object_Reference