CYBER SECURITY INTERNSHIP
TASK 1 REPORT

Task Title: Web Application Security Testing

Track code: FUTURE_CS_01

Intern Name: Eryl Bwiru Makorre

Aim:

The primary objective of this task is to perform an in-depth security assessment of a web application using the Damn Vulnerable Web Application (DVWA) platform in a controlled lab environment. The focus is on identifying and exploiting widely known web vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and authentication flaws through simulated brute-force attacks. This assessment aims to understand the potential impact of these vulnerabilities and provide strategic recommendations and best practices to enhance the overall security posture of modern web applications.

DVWA SQL Injection Vulnerability Exploitation

Target Environment:

- **Application:** DVWA (Damn Vulnerable Web Application)
- **Security Level:** Low
- **SQL Database:** MySQL
- **Module:** SQL Injection

Objective:

To demonstrate and document SQL Injection attacks on a vulnerable input field in DVWA to:

1. Retrieve user data.
2. List database tables.
3. Enumerate table columns.

Setting Up DVWA in Kali Linux:

Requirements:

- Apache web server
- MySQL/MariaDB
- PHP
- DVWA source code (from GitHub)

Login to DVWA

URL: http://localhost/DVWA/login.php

Default Credentials:

Username: admin **Password:** password

DVWA is Ready!

You can now start testing vulnerabilities like: SQL Injection, XSS, CSRF

Basic SQL Injection Test

- **Input: 1**

- **Result:**
  Displays record for ID: 1:

  First name: admin
  Surname: admin

- **Observation:**
  The backend executes:

  SELECT * FROM users WHERE id = '1'

  This confirms the input is being inserted directly into a SQL query without sanitization.

Screenshot: sql_1.png

Bypass Authentication & Dump All Users

- **Input:**

1' or '1'='1

- **Result:**

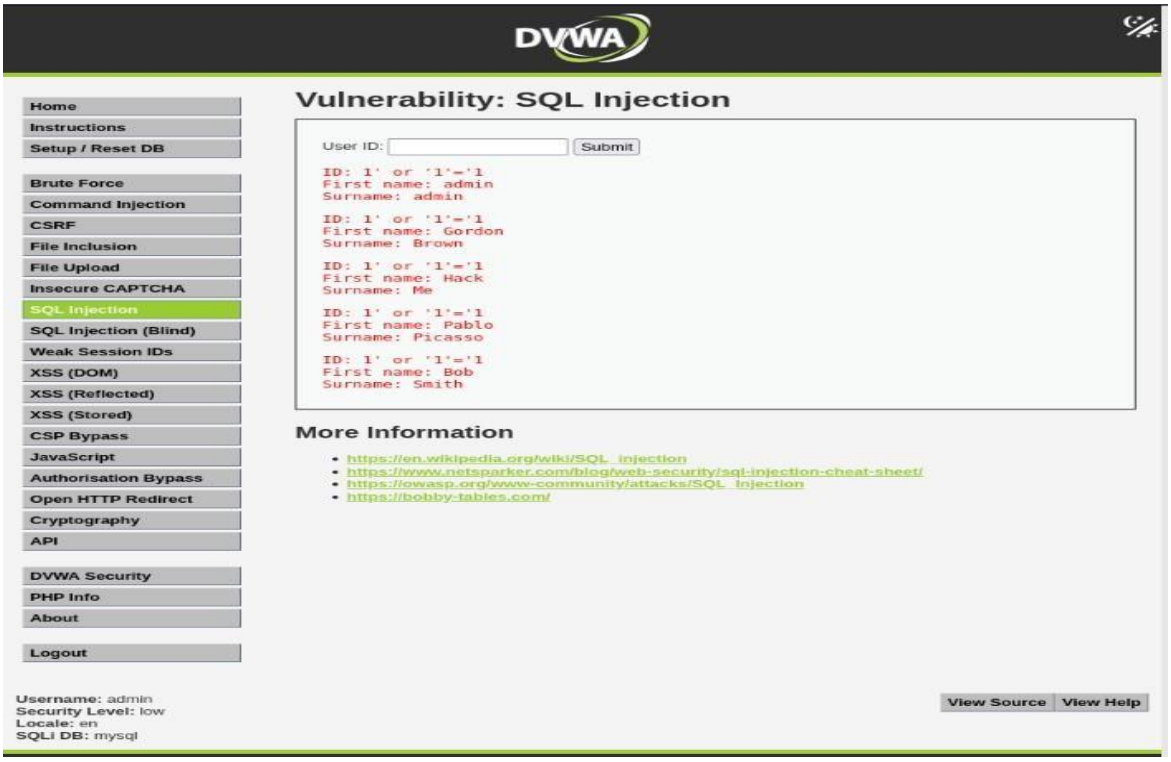
  Dumps all users from the database:

  Admin, Gordon Brown, Hack Me, Pablo Picasso, Bob Smith

- **Underlying Query:**

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

- **Impact:**
  - Authentication bypass
  - Full table enumeration
  - Critical breach of confidentiality



Screenshot: sql_2.png

Extract Table Names

- **Input:**

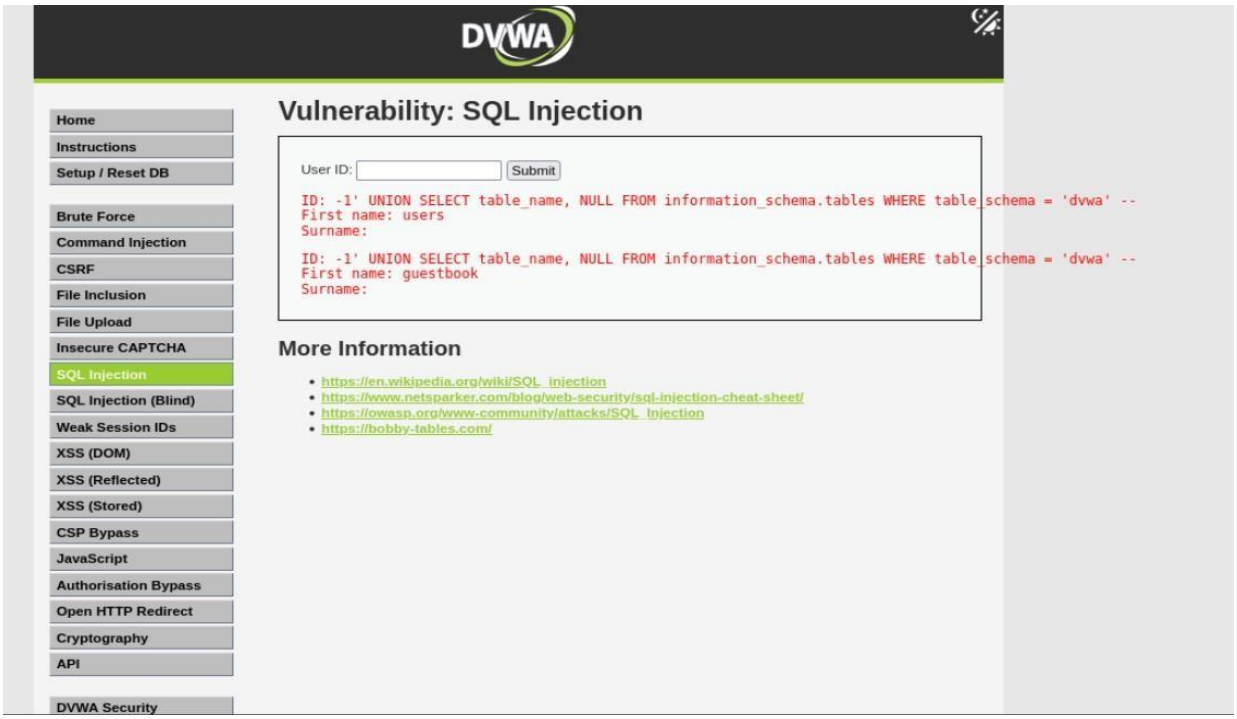
  -1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa' –

- **Result:**

  Reveals the following tables:
  - Guestbook
  - Users

- **Impact:**

  - Database schema enumeration
  - Attackers now know which tables to target next



Screenshot: sql_3.png

Extract Column Names from users Table

- **Input:**

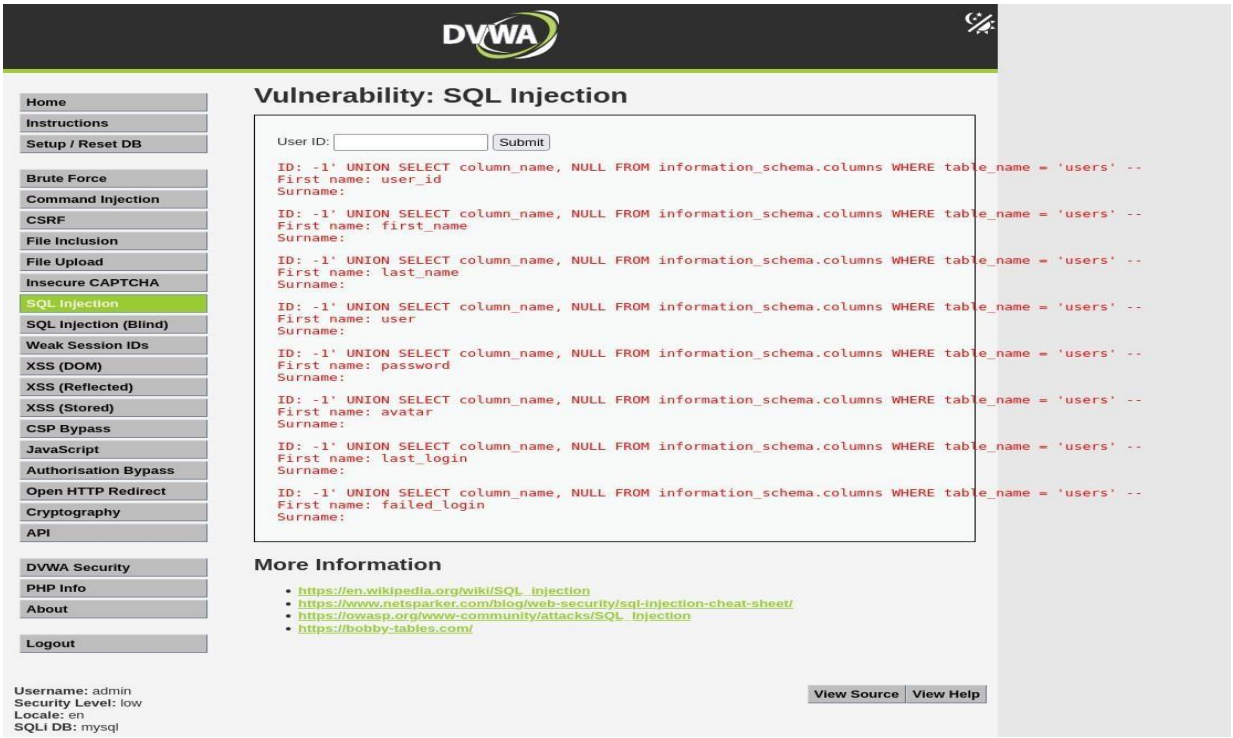
  -1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users' –

- **Result:**

  Enumerates column names:
  - user_id
  - first_name
  - Sur_name

- **Impact:**

  Full mapping of user table columns, including sensitive fields like:
  - First name of all the users

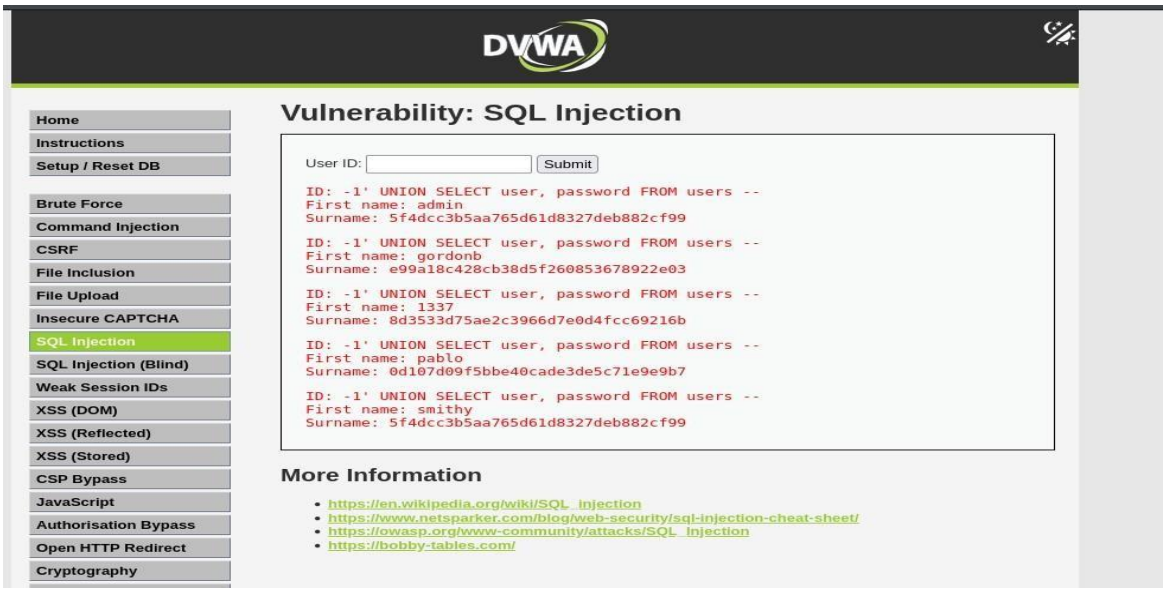


Screenshot: sql_4.png

Extract Usernames and Passwords from users Table

- **Input:**

  -1' UNION SELECT user, password FROM users –



Screenshot sql_5.png

- **Result:**

  Enumerates sensitive user credentials:

  - admin | 5f4dcc3b5aa765d61d8327deb882cf99
  - gordonb | e99a18c428cb38d5f260853678922e03
  - pablo | 0d107d09f5bbe40cade3de5c71e9e9b7

- **Impact:**

  Full compromise of authentication data:

  - Reveals usernames of all registered users
  - Leaks MD5-hashed passwords, which can be cracked easily
  - Enables unauthorized login and privilege escalation (e.g., admin access)

What was done: SQL injection techniques were applied to extract sensitive data by retrieving user records, enumerating database tables and columns, and capturing usernames with their corresponding hashed passwords.

Conclusion: Through SQL injection, attackers were able to retrieve complete user data and database schema details. This exploitation revealed the absence of essential security measures like input validation and the failure to use parameterized SQL queries.

DVWA XSS Vulnerability Testing

Overview

This report outlines the identification and exploitation of three types of Cross-Site Scripting (XSS) vulnerabilities using DVWA:

- Reflected XSS
- Stored XSS
- DOM-Based XSS

---

1. Reflected XSS

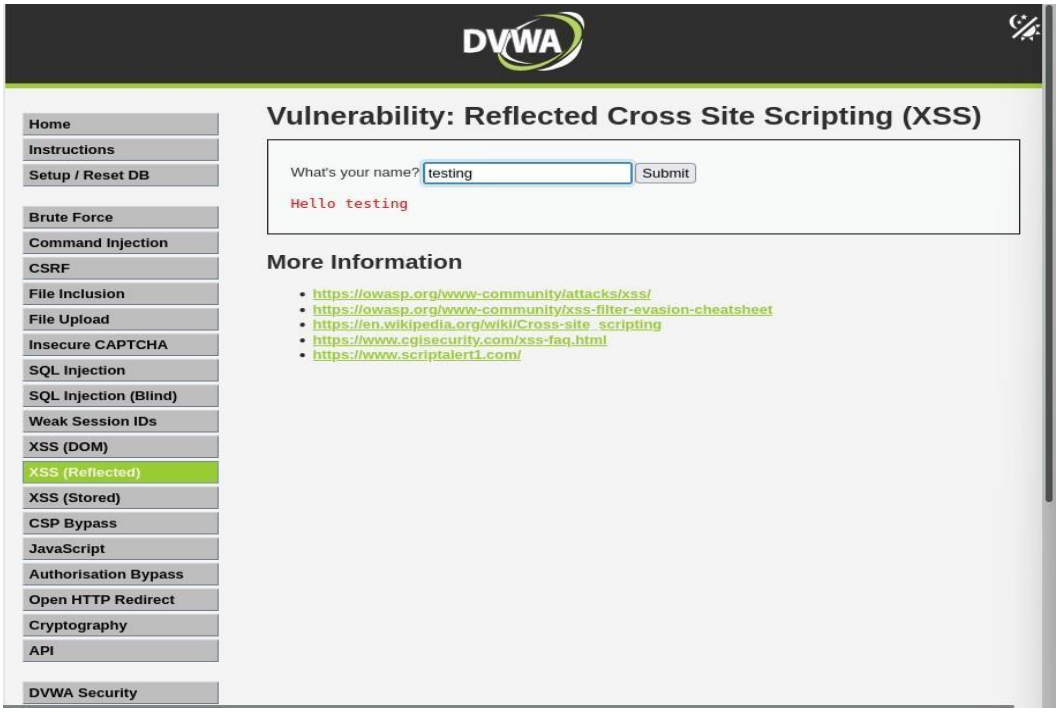
Description:
Reflected XSS occurs when user input is reflected immediately in the page response without proper validation or encoding.

Steps Taken:

- Navigated to the "XSS (Reflected)" module.
- Entered the string `testing` to test input reflection.
- Injected payload: `<script>alert('XSS')</script>`

Result:

- Payload executed immediately in the browser.



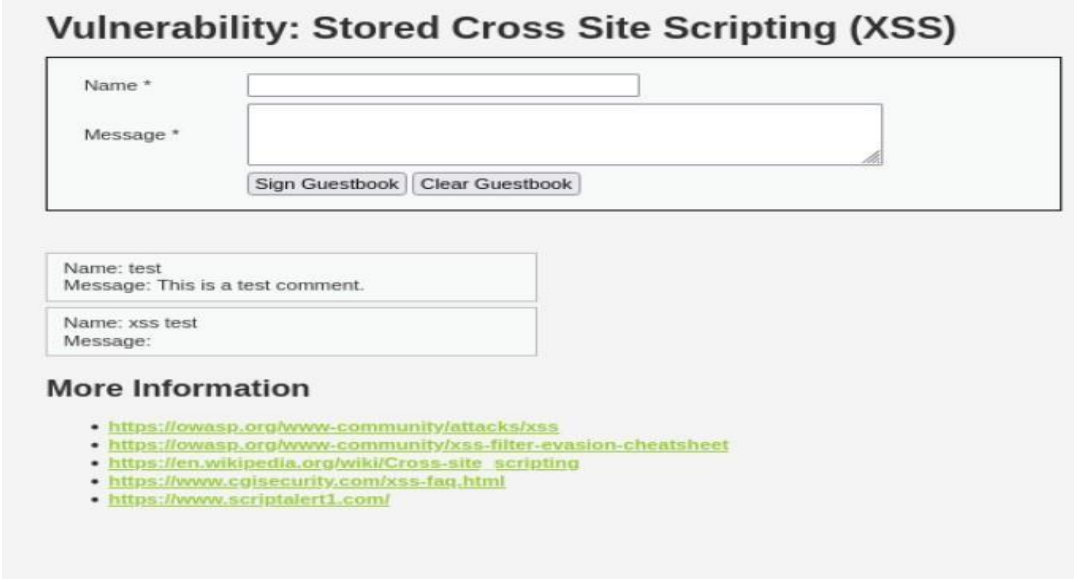
Screenshot: xssr_1.png

2. Stored XSS

Description:
Stored XSS involves injecting a malicious script that gets saved on the server (e.g., in a database) and executes whenever the data is viewed.

Steps Taken:

- Navigated to "XSS (Stored)" module.
- Input Name: `xss test`
- Input Message: `<script>alert(1)</script>`



Screenshot: xsss_2.png

- Submitted the form.

Result: Alert box popped up when the stored message was rendered.

Screenshot: xsss_3.png

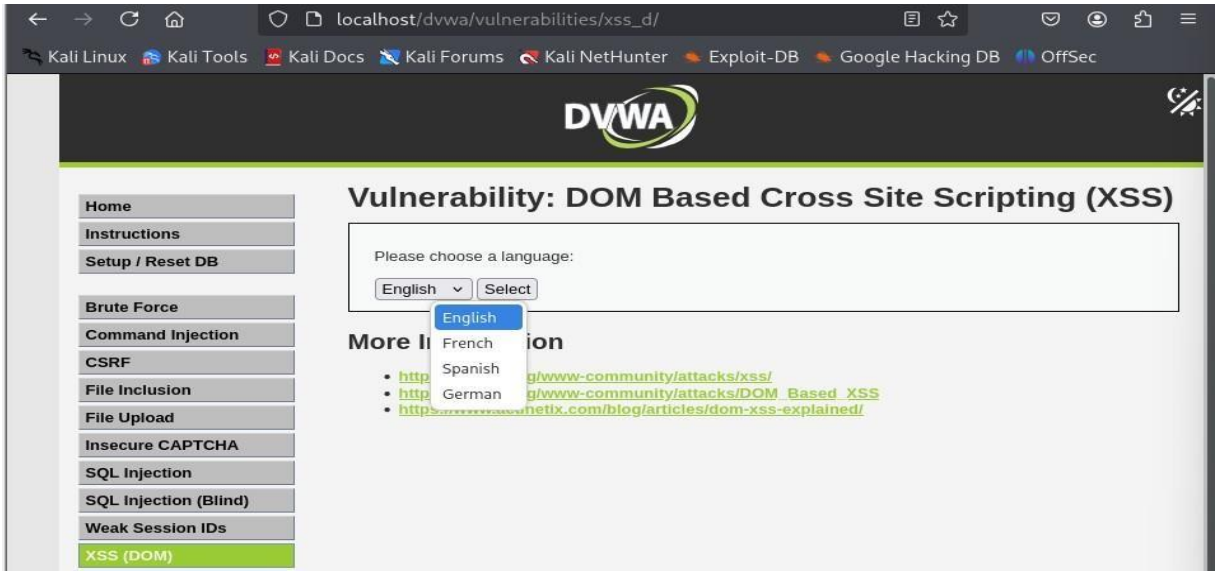
3. DOM-Based XSS

Description:

DOM-Based XSS occurs on the client-side where JavaScript manipulates the DOM using untrusted data (e.g., URL parameters).

Steps Taken:

- Navigated to "XSS (DOM)" module.

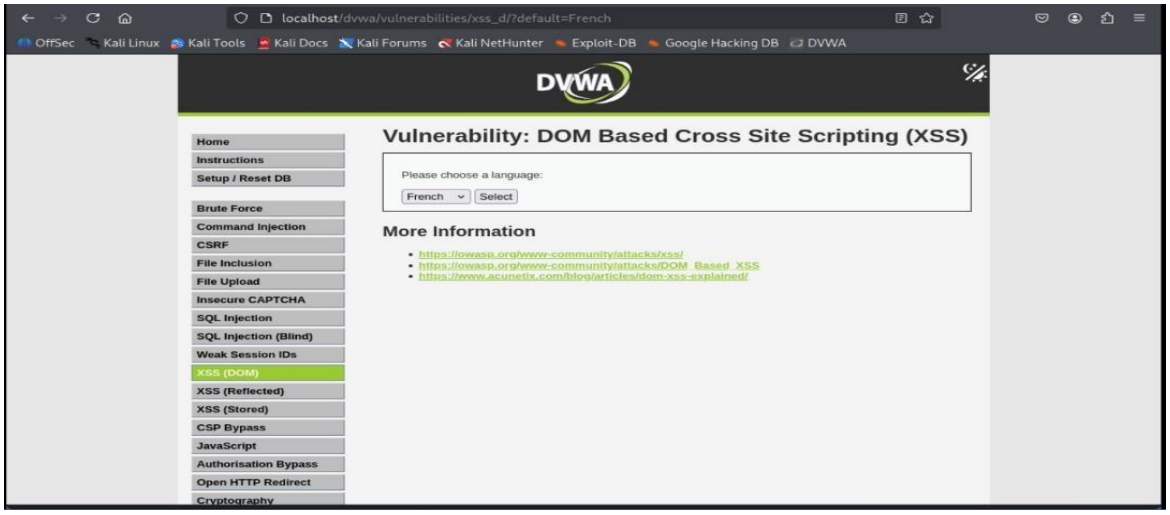


Screenshot: xssd_1.png

- Modified URL to: ?default=French and observed behavior.

Result:

- JavaScript executed due to lack of sanitization in JavaScript code.



Screenshot: xssd_2.png

Summary Table:

XSS Type	Stored on Server	Trigger Location	Risk Level	Example Field
Reflected XSS	No	Immediate response	Medium	Search, Forms
Stored XSS	Yes	On viewing saved data	High	Comments, Messages
DOM-Based XSS	No	Client-side script code	High	URL Parameters

Conclusion: All three types of XSS—Reflected, Stored, and DOM-Based—were successfully exploited, revealing how inadequate input validation and missing output encoding can result in serious client-side vulnerabilities.

Brute Force Attack Simulation on DVWA using Burp Suite

Objective:

To simulate a brute force attack on a vulnerable web application and understand how attackers exploit weak authentication mechanisms using Burp Suite's Intruder feature.

Tools Used:

- Burp Suite
- DVWA (Damn Vulnerable Web Application)
- Kali Linux
- Custom Password Wordlist

Procedure:

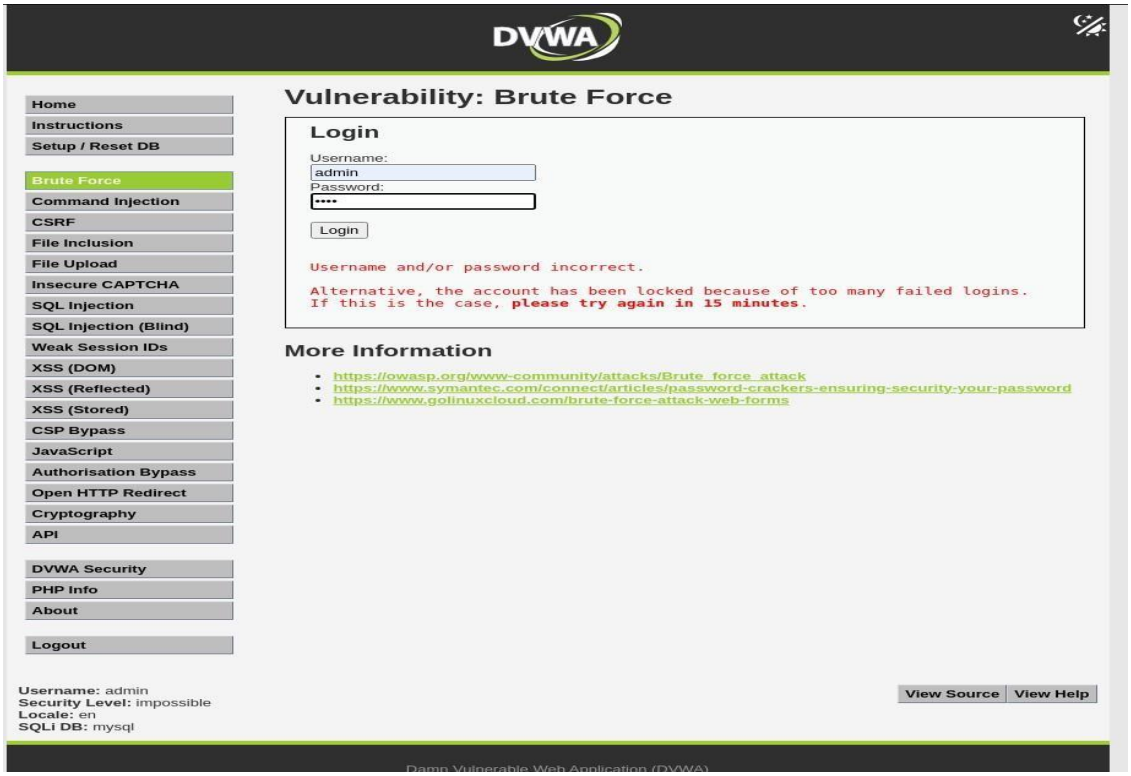
1. Setup:

- DVWA configured to 'Impossible' difficulty level
- Logged into DVWA with valid admin credentials

Screenshot: BF_1.png

2. Target Identification:

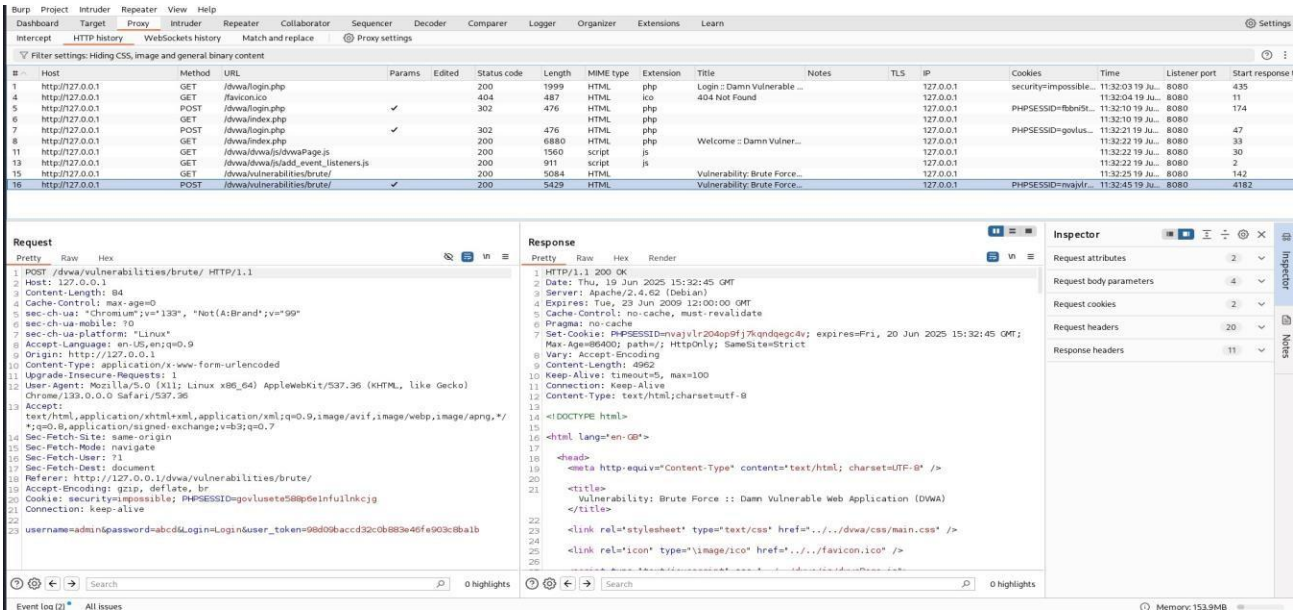
- Navigated to the "Brute Force" module in DVWA
- Attempted a failed login to capture the POST request



Screenshot: BF_2.png

3. Intercept & Analyze Request:

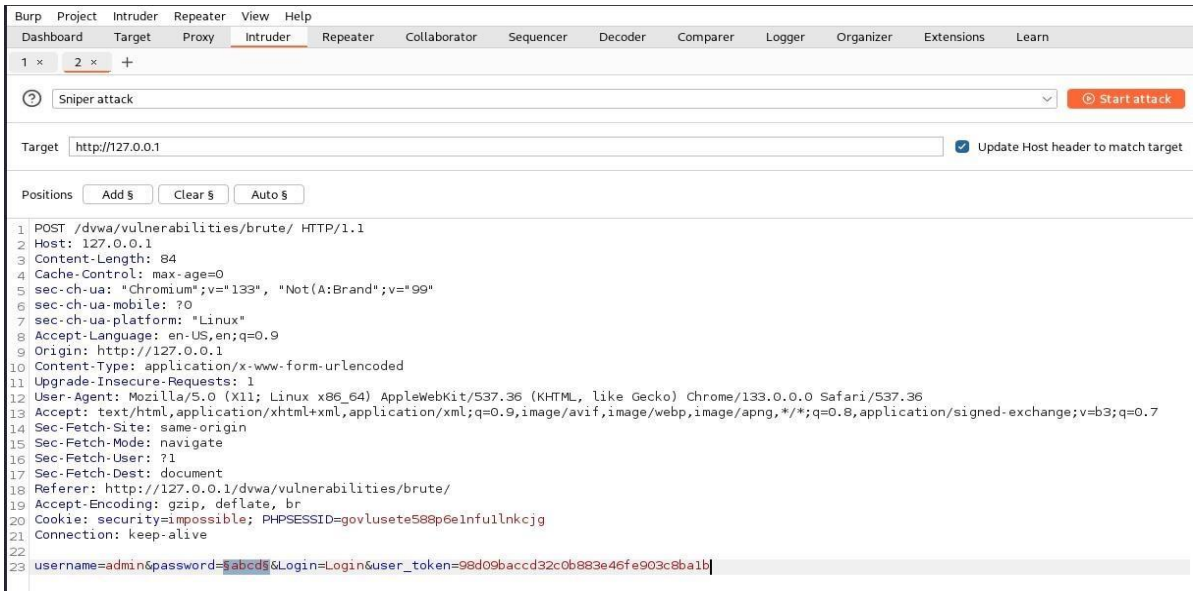
- Opened Burp Suite and enabled the Intercept option
- Captured the login POST request and sent it to Intruder



Screenshot: BF_3.png

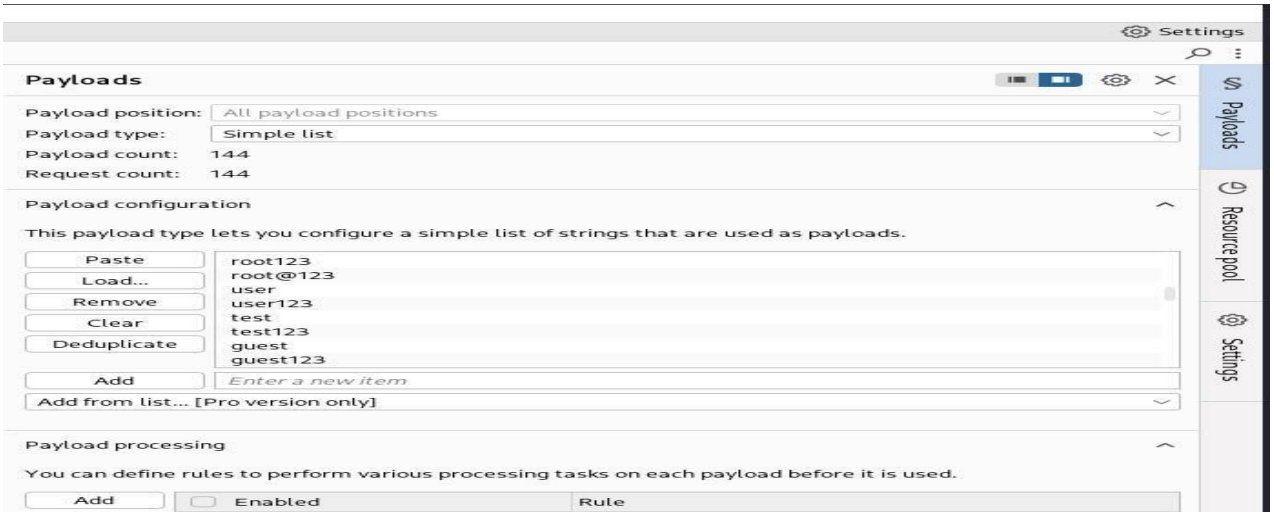
4. Configure Burp Intruder:

- Set payload position for the password field



Screenshot: BF_4.png

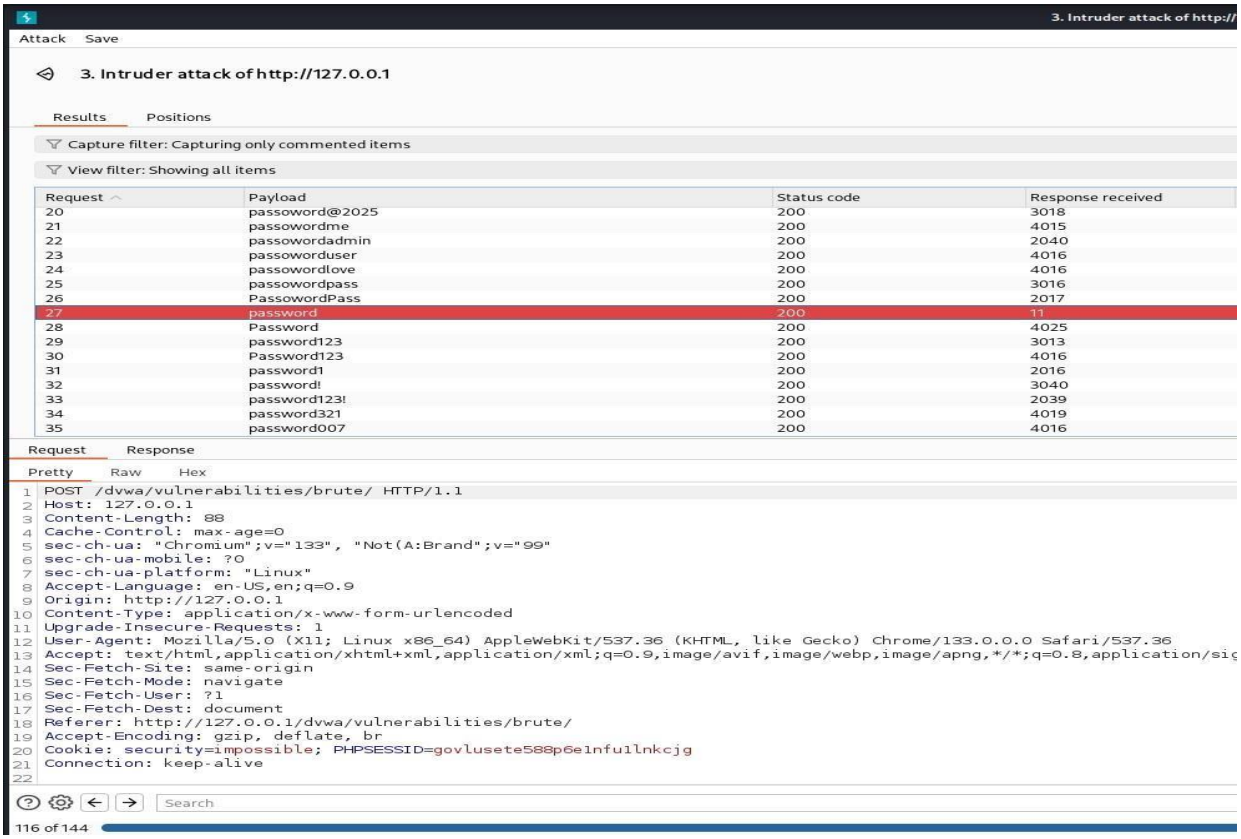
- Loaded a custom wordlist of potential passwords



Screenshot: BF_5.png

5. Execute Attack:

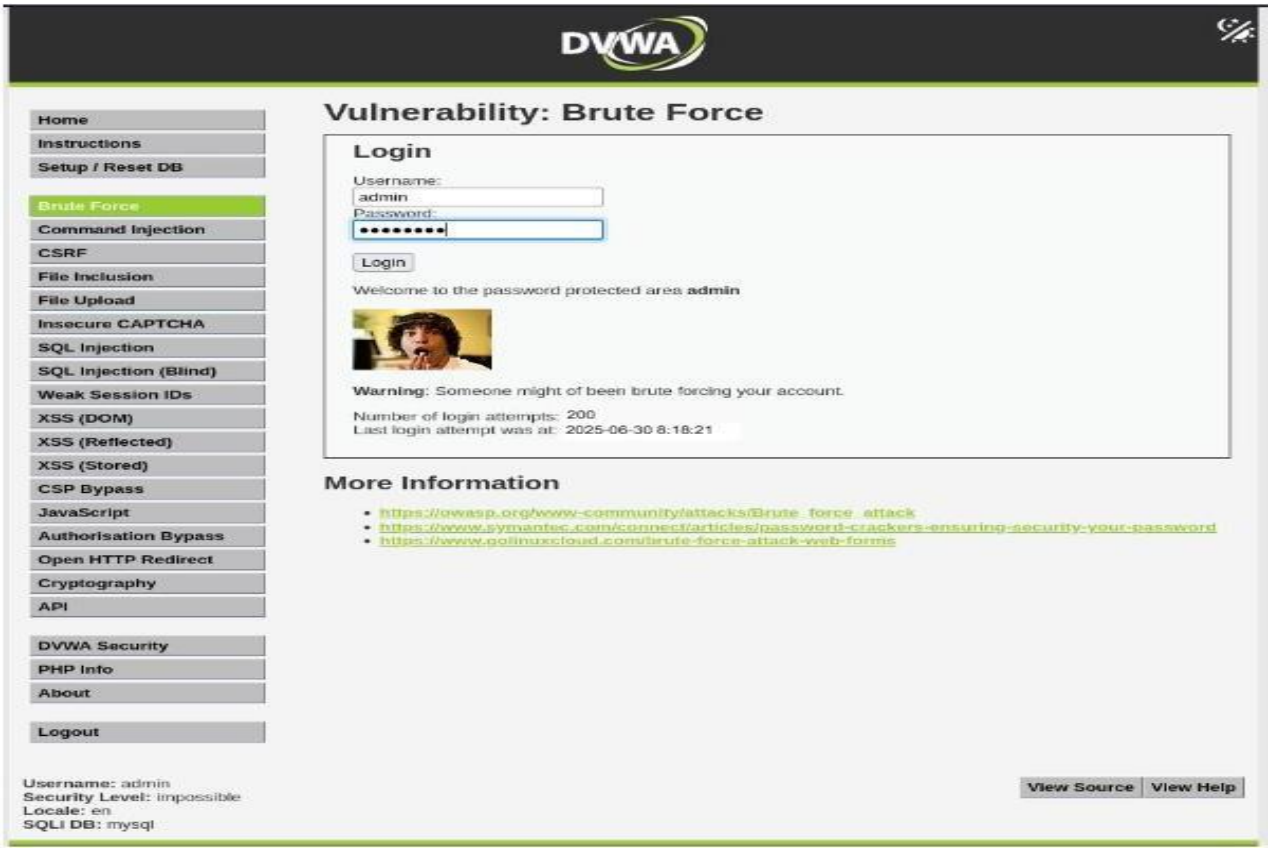
- Launched attack via Burp Intruder
- Monitored response lengths and status codes
- Identified correct password based on unique response behavior



Screenshot: BF_6.png

Result:

Successful login detected using the password: **password**



Screenshot: BF_7.png

Analysis & Observations:

- Response codes and content length differences were key in detecting a successful login
- Lack of rate-limiting or CAPTCHA made the attack feasible even at the highest security level

Security Recommendations:

- Implement account lockout after multiple failed attempts
- Enforce CAPTCHA to mitigate automated attacks
- Apply strong password policies and multi-factor authentication

---

Learning Outcomes:

- Deepened understanding of brute force attack methodologies
- Enhanced skills in HTTP request analysis and manipulation
- Practical exposure to Burp Suite's Intruder module

Ethical Note:
All actions were performed in a safe, controlled lab environment using intentionally vulnerable software for educational purposes only. Unauthorized access to real systems is illegal and unethical.

Brute Force Authentication Attack

- **What was done:** A brute-force attack was simulated using Burp Suite Intruder with a custom wordlist.
- **Conclusion:** The application lacked protections like rate limiting, CAPTCHA, or lockout mechanisms, making it vulnerable even on "Impossible" security settings. This underscores the necessity of robust authentication defense mechanisms.

Overall Conclusion:

The security evaluation carried out on DVWA effectively highlighted how prevalent web vulnerabilities—such as SQL Injection, Cross-Site Scripting (XSS), and insufficient authentication mechanisms—can be leveraged by attackers to compromise a web application. Through hands-on exploitation, each test exposed critical security weaknesses capable of leading to serious consequences including data leaks, unauthorized system access, and significant reputational and financial harm in real-world environments.

This practical assessment reinforces the urgent need for organizations to adopt a proactive and security-first development approach. To build resilient web applications, it is essential to:

- Enforce strict input validation and output encoding to prevent injection and scripting attacks.
- Follow secure coding standards, such as the use of parameterized queries to mitigate SQL injection risks.
- Implement robust authentication frameworks, including rate-limiting, CAPTCHA, and multi-factor authentication (MFA) to deter brute-force and automated attacks.
- Conduct regular security assessments, including penetration testing and code reviews, to detect and resolve vulnerabilities before they can be exploited.

In conclusion, this task not only demonstrated the ease with which insecure applications can be compromised but also emphasized the importance of continuous security awareness, developer training, and defensive programming in safeguarding web environments.

References:

[1] D. Hunt, *Damn Vulnerable Web Application (DVWA)*. GitHub. [Online]. Available: https://github.com/digininja/DVWA [Accessed: Jul. 14, 2025].

[2] OWASP Foundation, *Cross Site Scripting (XSS)*. OWASP. [Online]. Available: https://owasp.org/www-community/attacks/xss/ [Accessed: Jul. 14, 2025].

[3] PortSwigger Ltd., *Burp Suite Documentation*. PortSwigger. [Online]. Available: https://portswigger.net/burp/documentation [Accessed: Jul. 14, 2025].