CYBER SECURITY INTERNSHIP - TASK 1 REPORT

Task Title :- WEB APPLICATION SECURITY TESTING

Track_code :- FUTURE_CS_01

Intern Name :- Yandrapu Mahesh

Aim :-

To conduct comprehensive web application security testing using the DVWA (Damn Vulnerable Web Application) platform in a controlled environment, focusing on identifying and exploiting common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Authentication Weaknesses through simulated brute force attacks. The objective is to analyze the impact of these attacks and propose best practices for securing 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

1️⃣ Basic SQL Injection Test

- **Input:**

```
1
```

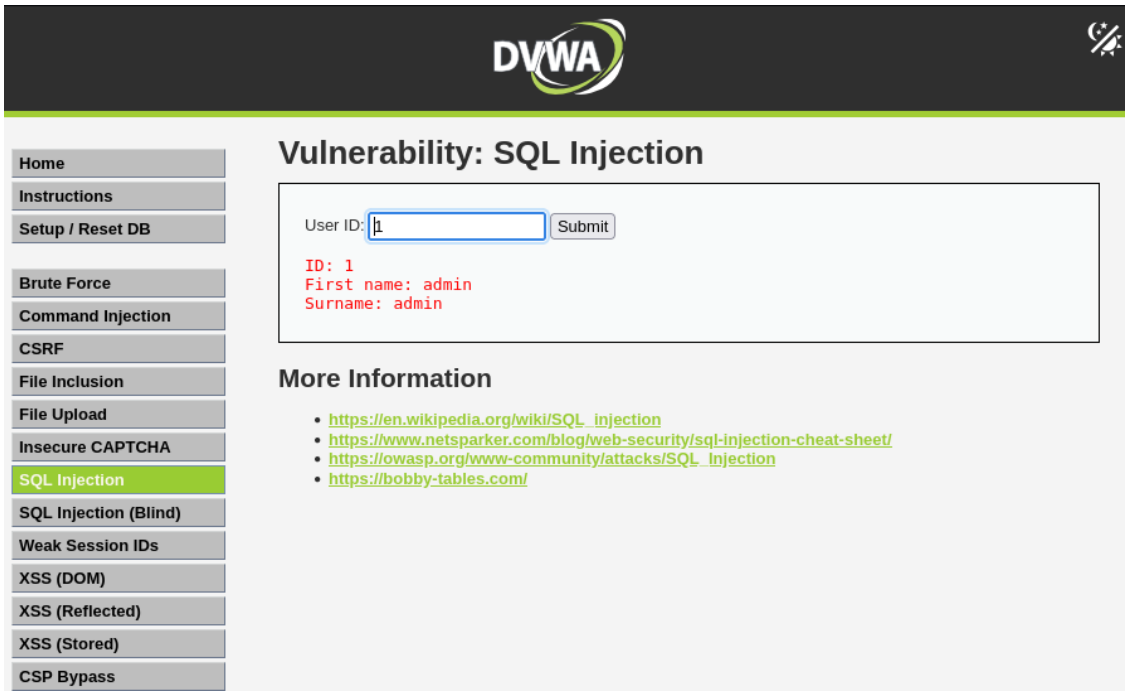
- **Result:**
  Displays record for `ID: 1`:

```
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: 1.png

---

2 Bypass Authentication & Dump All Users

- **Input:**

1' or '1'='1

- **Result:**

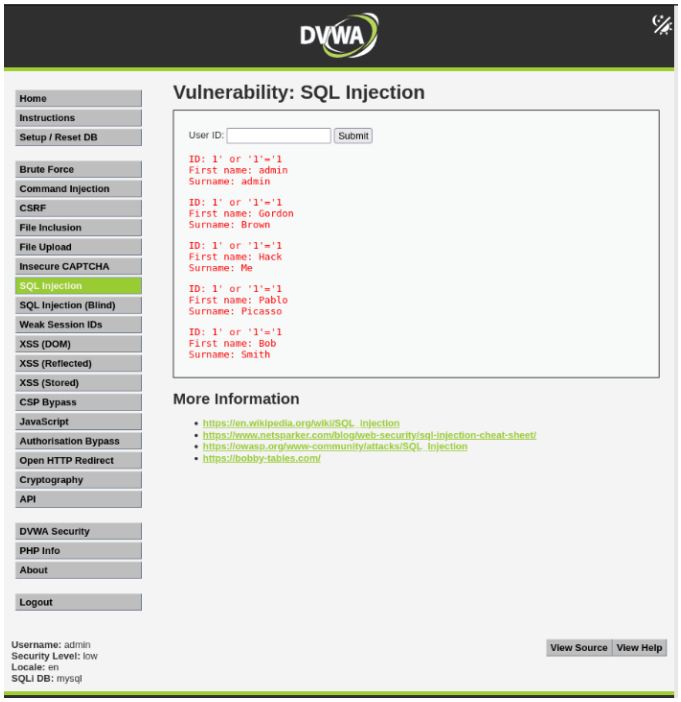
Dumps all users from the database:

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: 2.png

---

3 Extract Table Names

- **Input:**

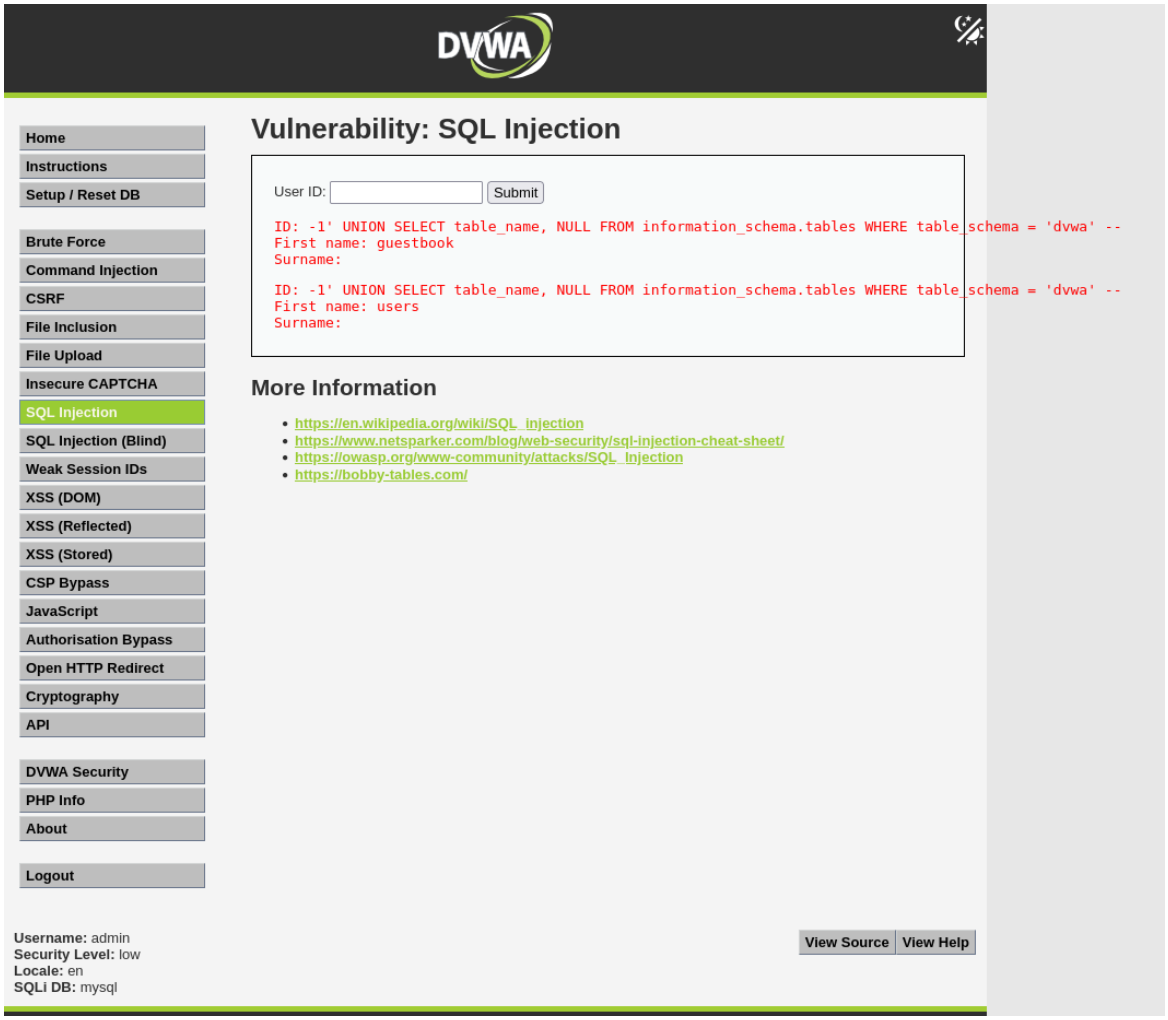
  -1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa' –

- **Result:**

  Reveals the following tables:
    - guestbook
    - users

- **Impact:**

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



Screenshot: 3.png

4 Extract Column Names from users Table

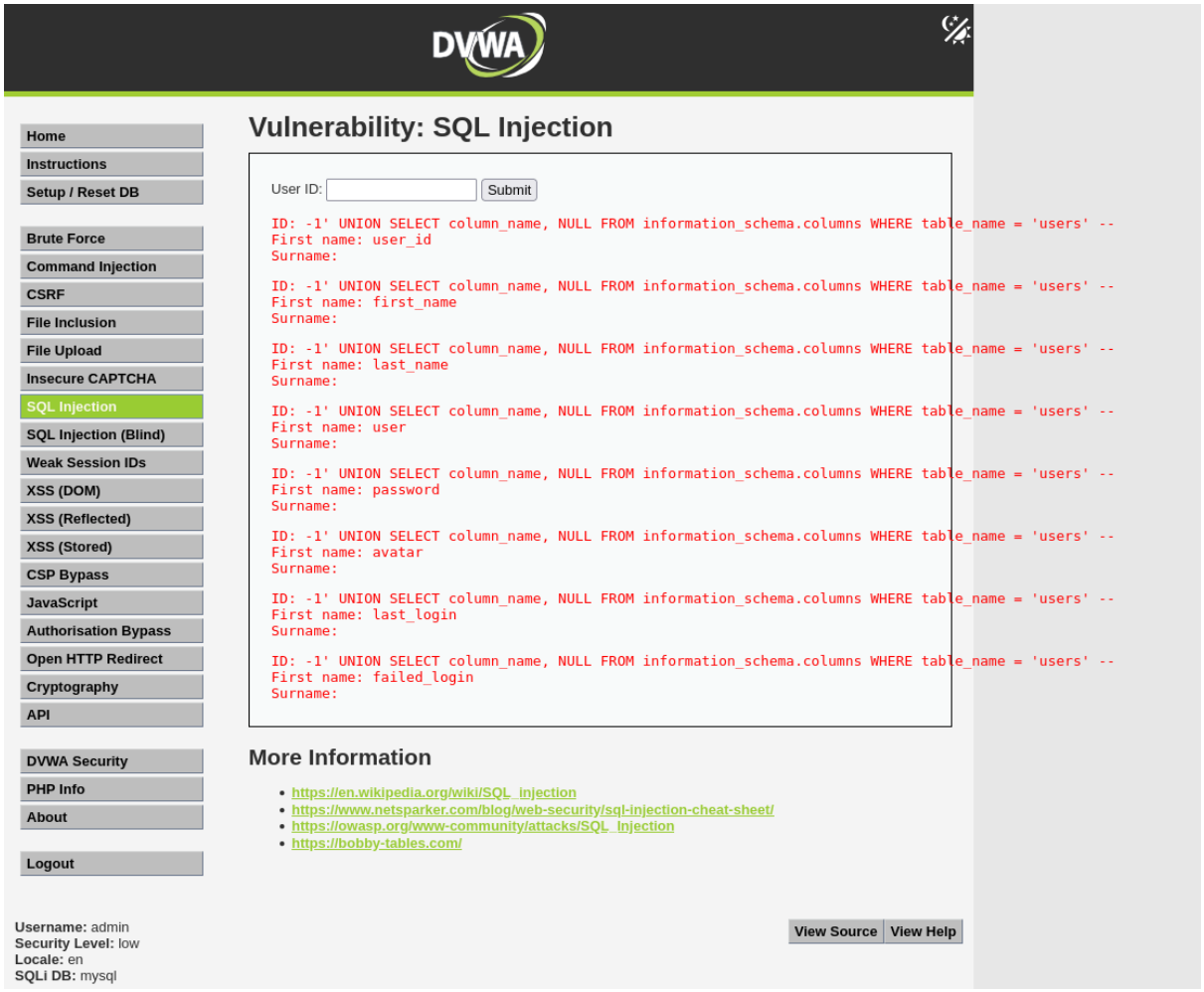
- **Input:**

  -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



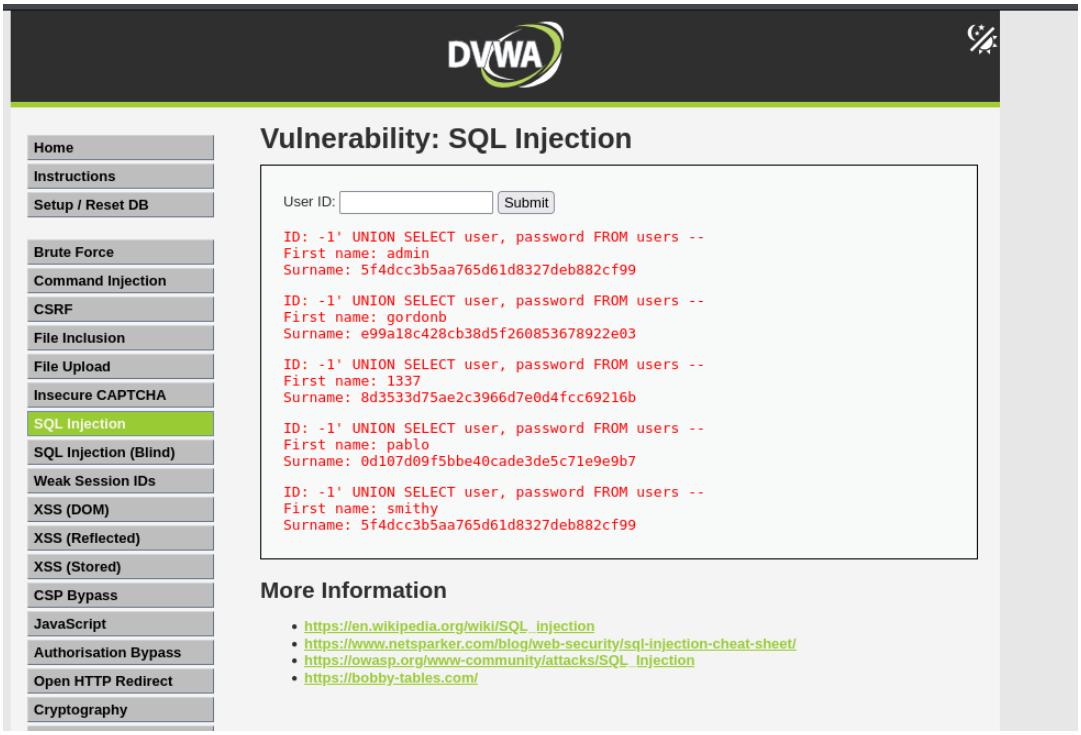
Screenshot: 4.png

---

5 Extract Usernames and Passwords from users Table

- **Input:**

  -1' UNION SELECT user, password FROM users –

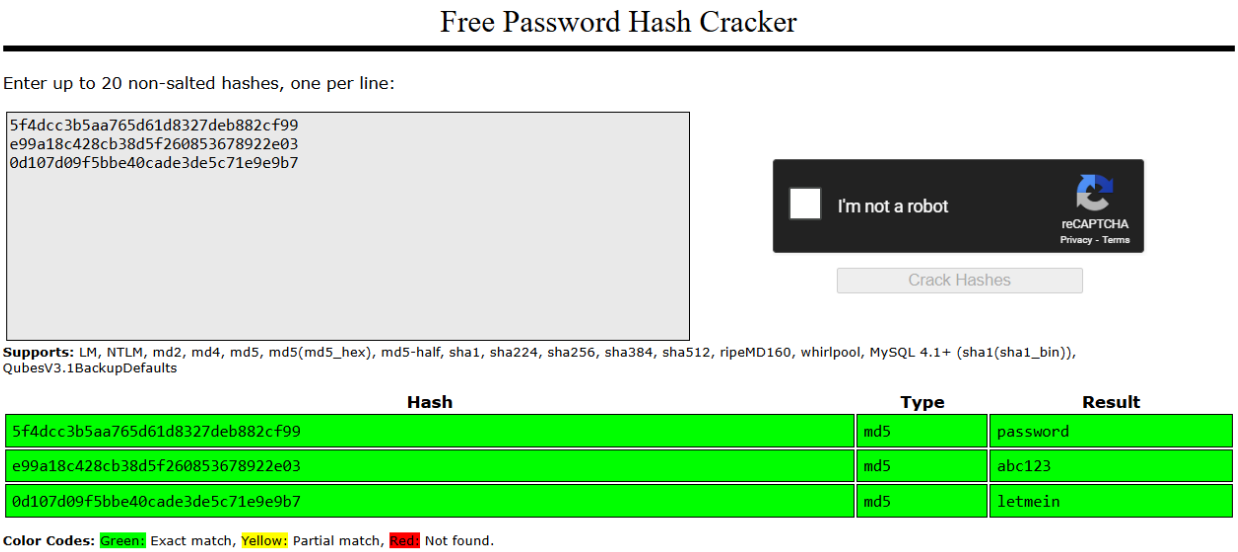


Screenshot 5.png

- **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: Multiple SQL injection payloads were used to retrieve data, enumerate tables/columns, and extract usernames and hashed passwords.

Conclusion: The SQL injection vulnerability allowed full access to sensitive user data and schema information. This highlights the absence of input sanitization and use of parameterized 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

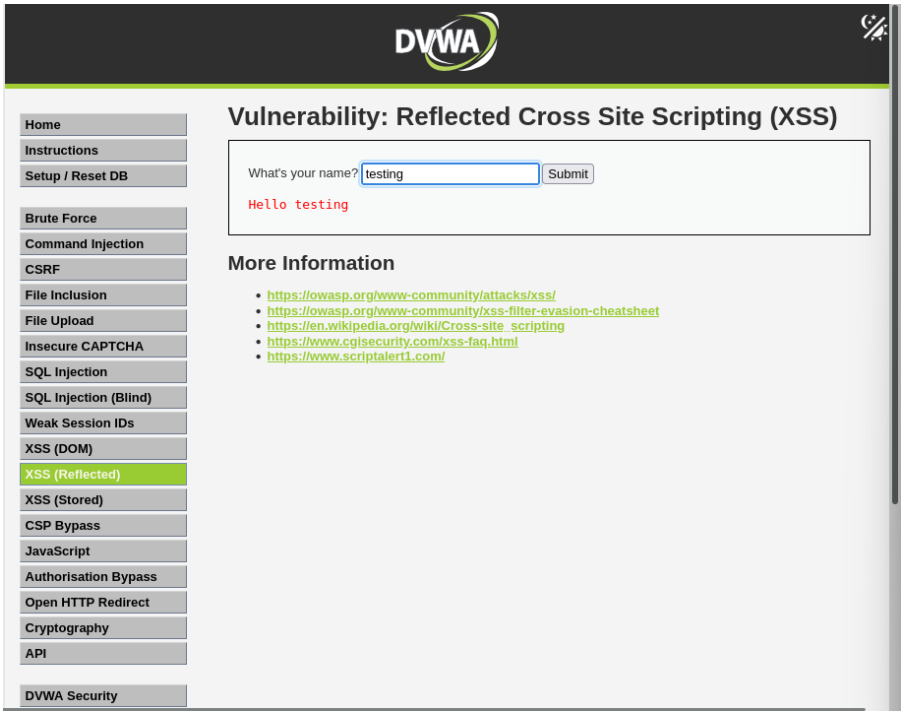
Description:
Reflected XSS occurs when user input is reflected immediately in the page response without proper validation or encoding.

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.

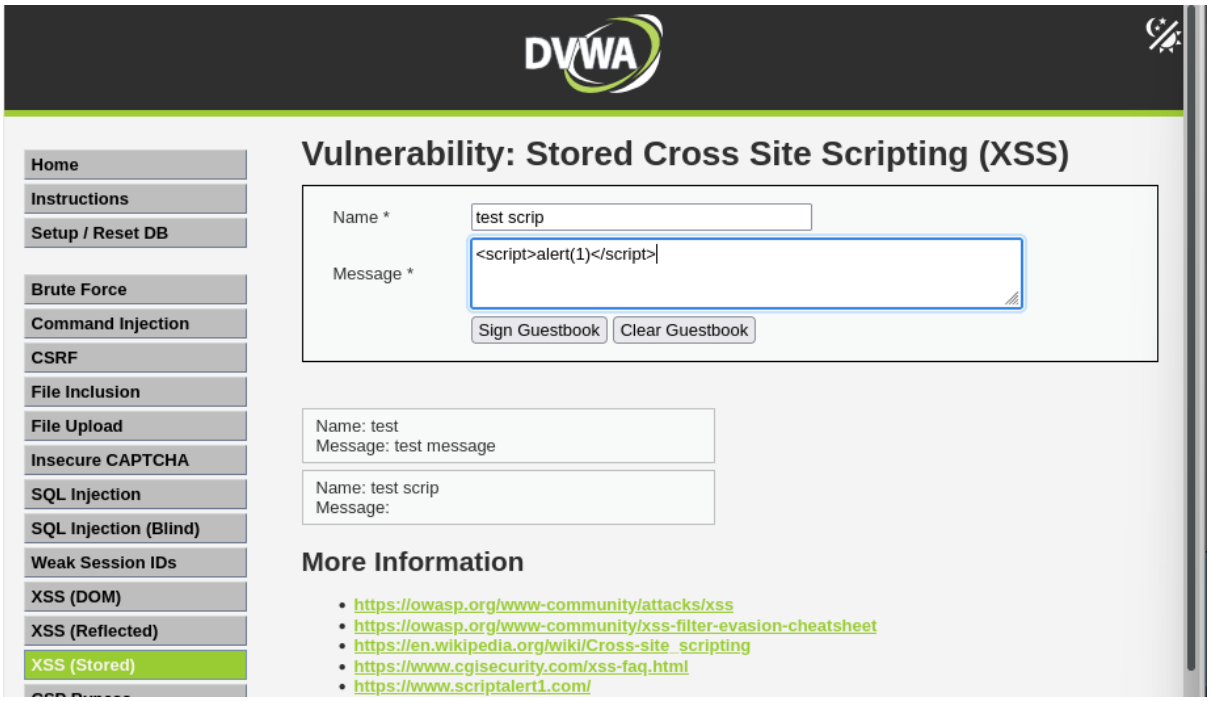


2. Stored XSS

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:

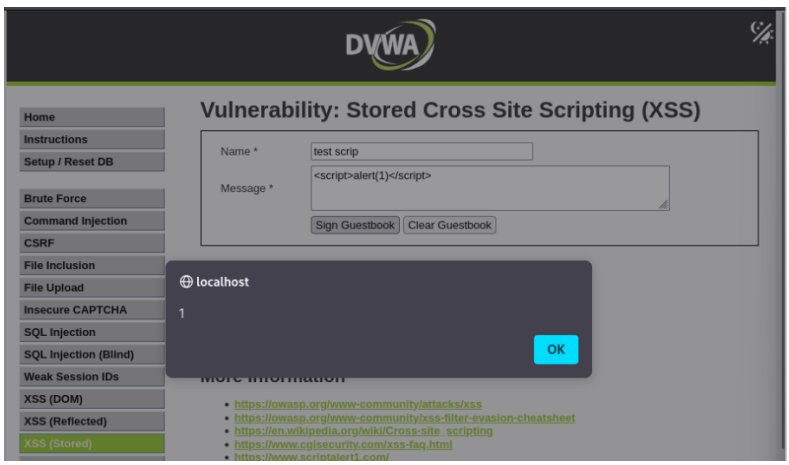
- Navigated to "XSS (Stored)" module.
- Input Name: `test scrip`
- Input Message: `<script>alert(1)</script>`



- Submitted the form.

Result:

- Alert box popped up when the stored message was rendered.

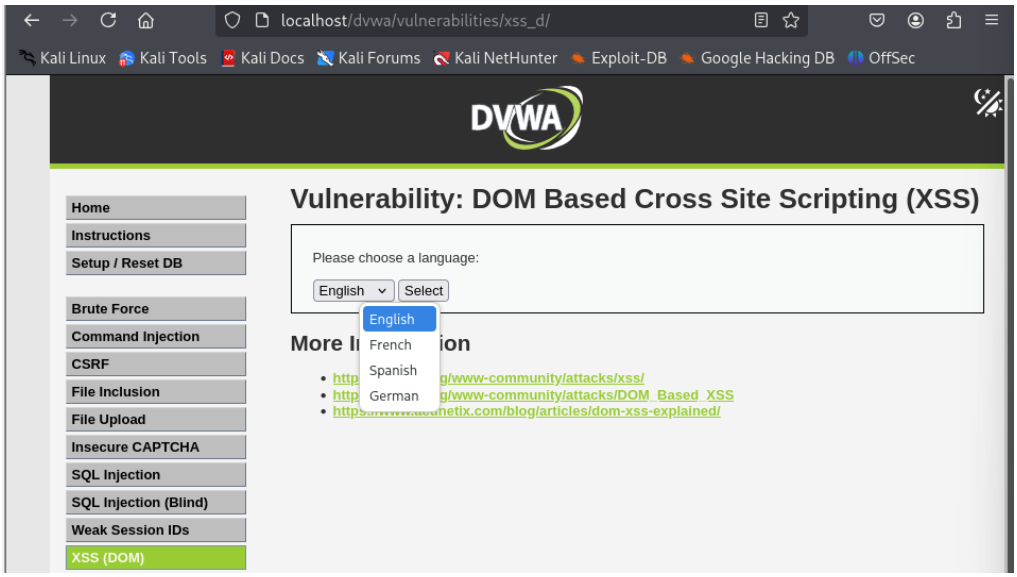


3. DOM-Based XSS

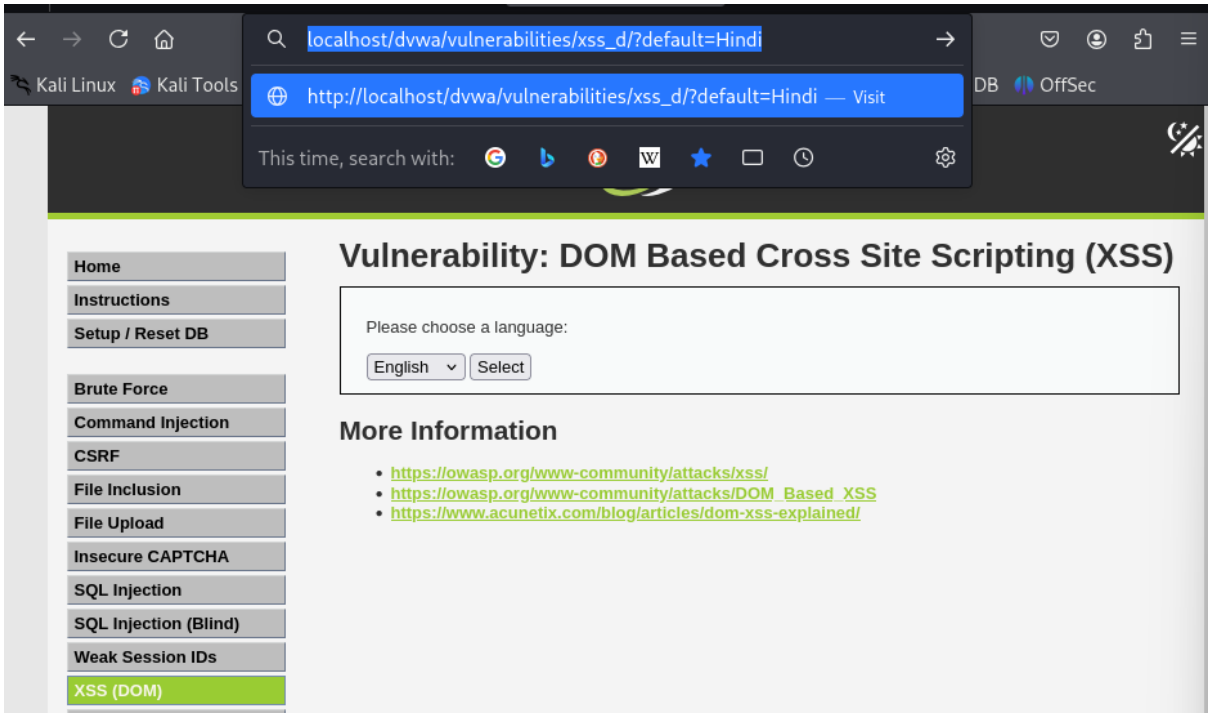
Description:
DOM-Based XSS occurs on the client-side where JavaScript manipulates the DOM using untrusted data (e.g., URL parameters).

Steps Taken:

- Navigated to "XSS (DOM)" module.

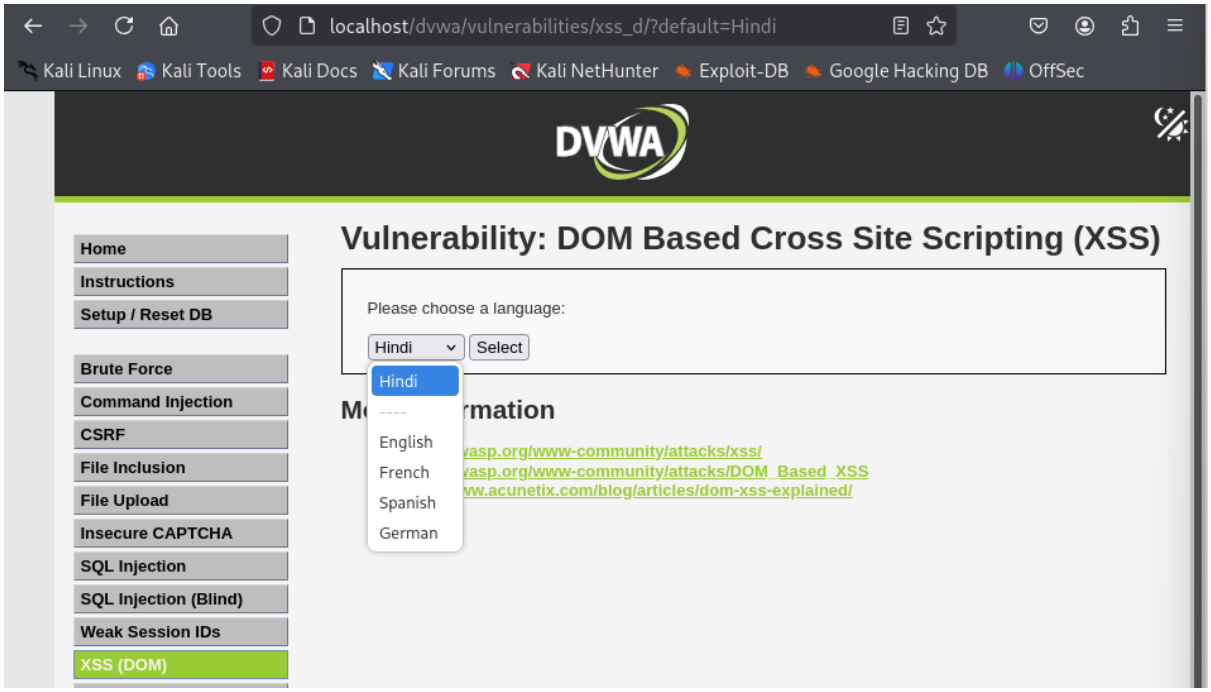


- Modified URL to: ?default=Hindi and observed behavior.



Result:

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



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 forms of XSS were exploitable, demonstrating how poor input validation and lack of output encoding can lead to severe client-side security issues.

Web Application Security Assessment

Environment: Kali Linux, DVWA (Difficulty: Impossible), Sample Web Application

---

🔐 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

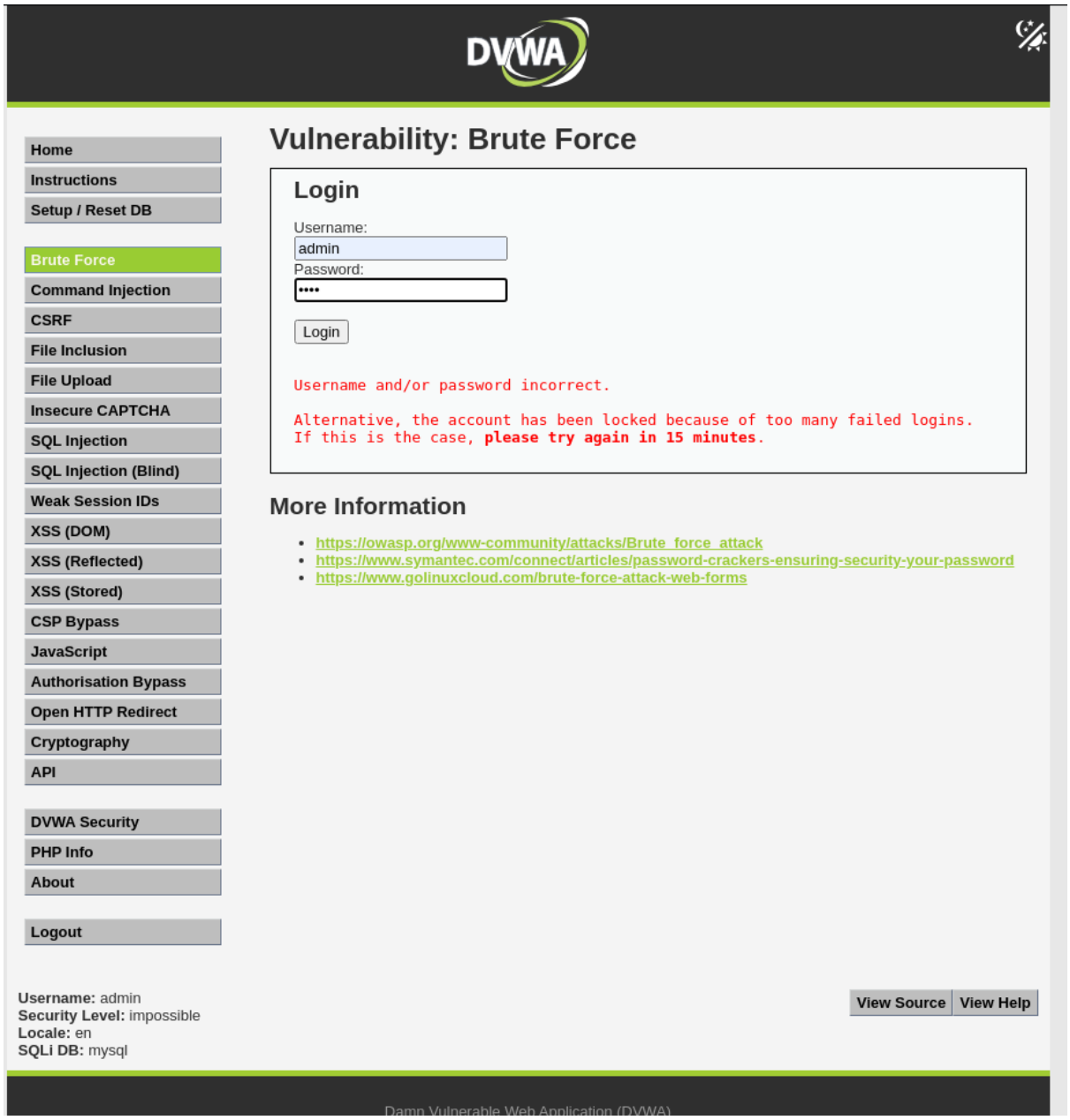
Procedure:

1. Setup:

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

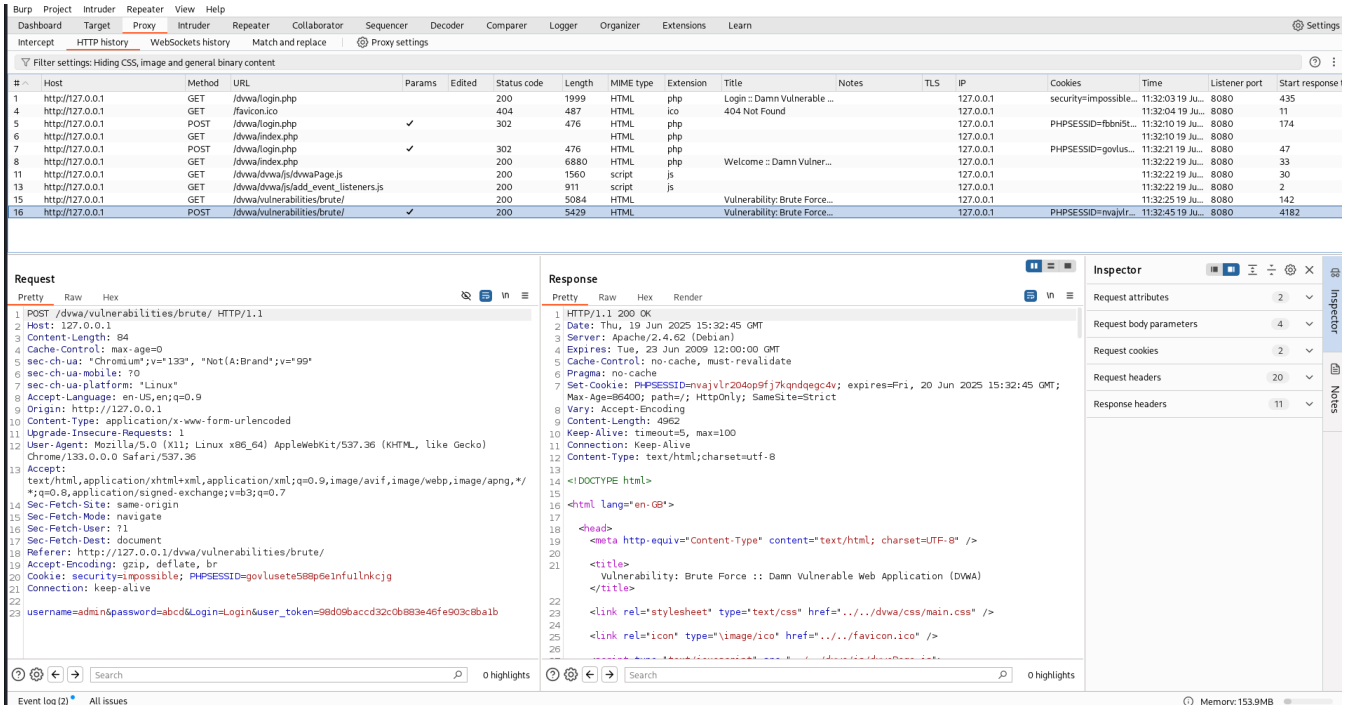
2. Target Identification:

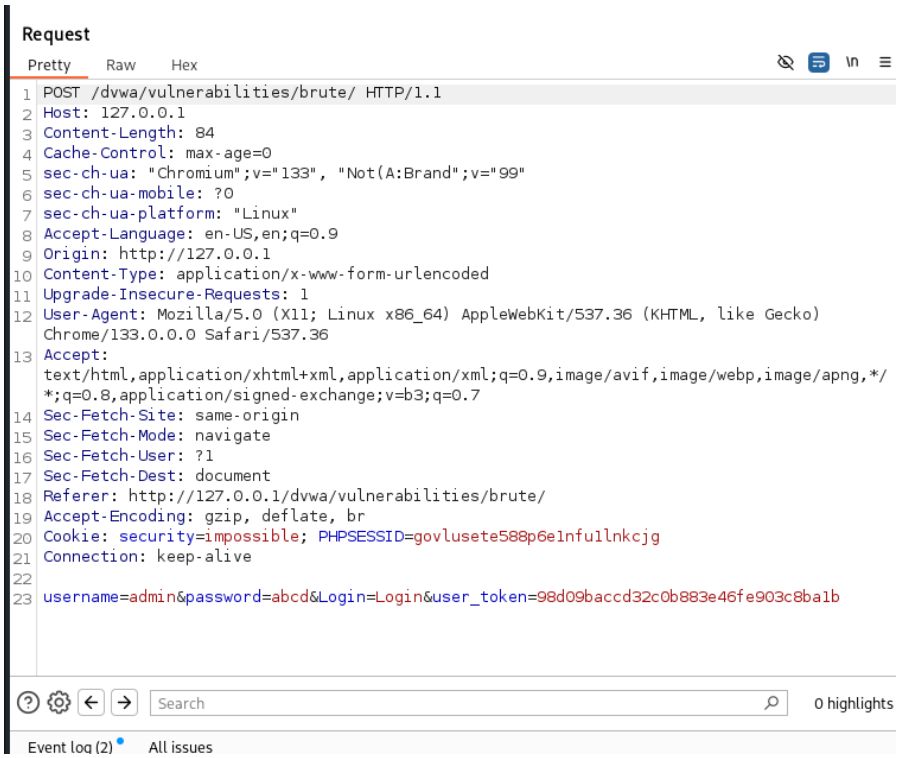
- Navigated to the "Brute Force" module in DVWA
- Attempted a failed login to capture the POST request



3. Intercept & Analyze Request:

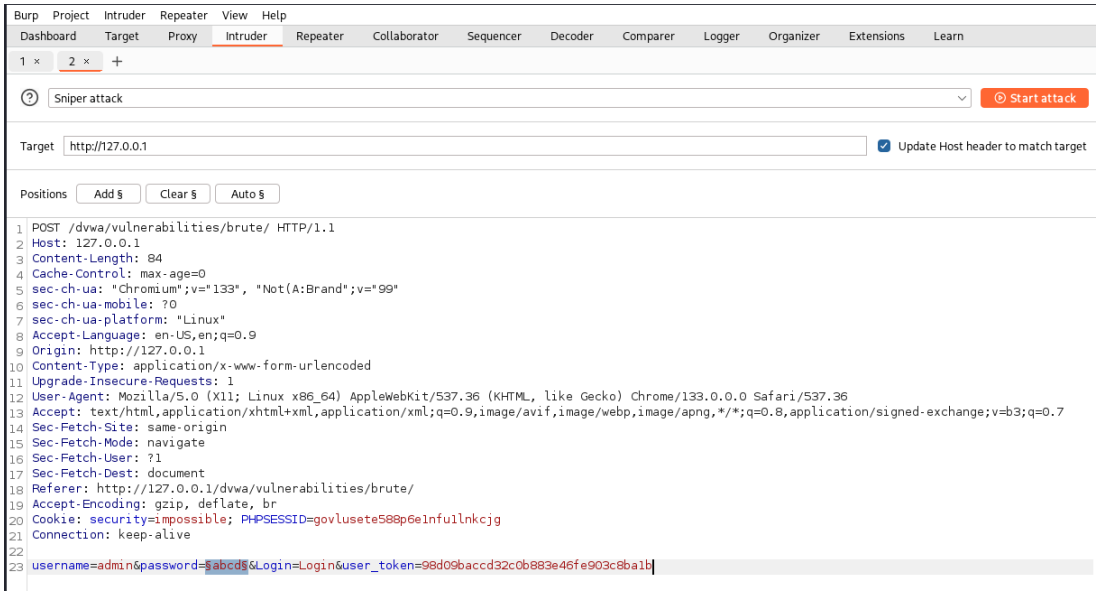
- Opened Burp Suite and enabled the Intercept option
- Captured the login POST request and sent it to Intruder



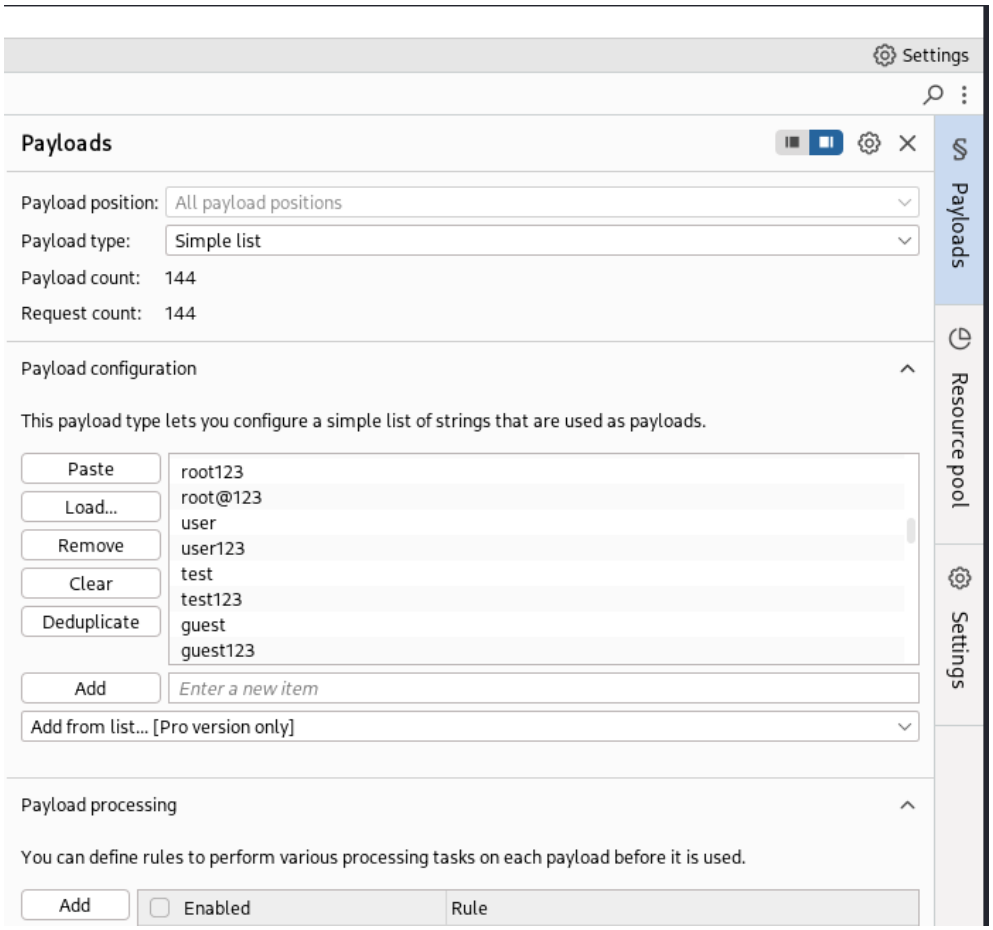


4. Configure Burp Intruder:

- Set payload position for the password field

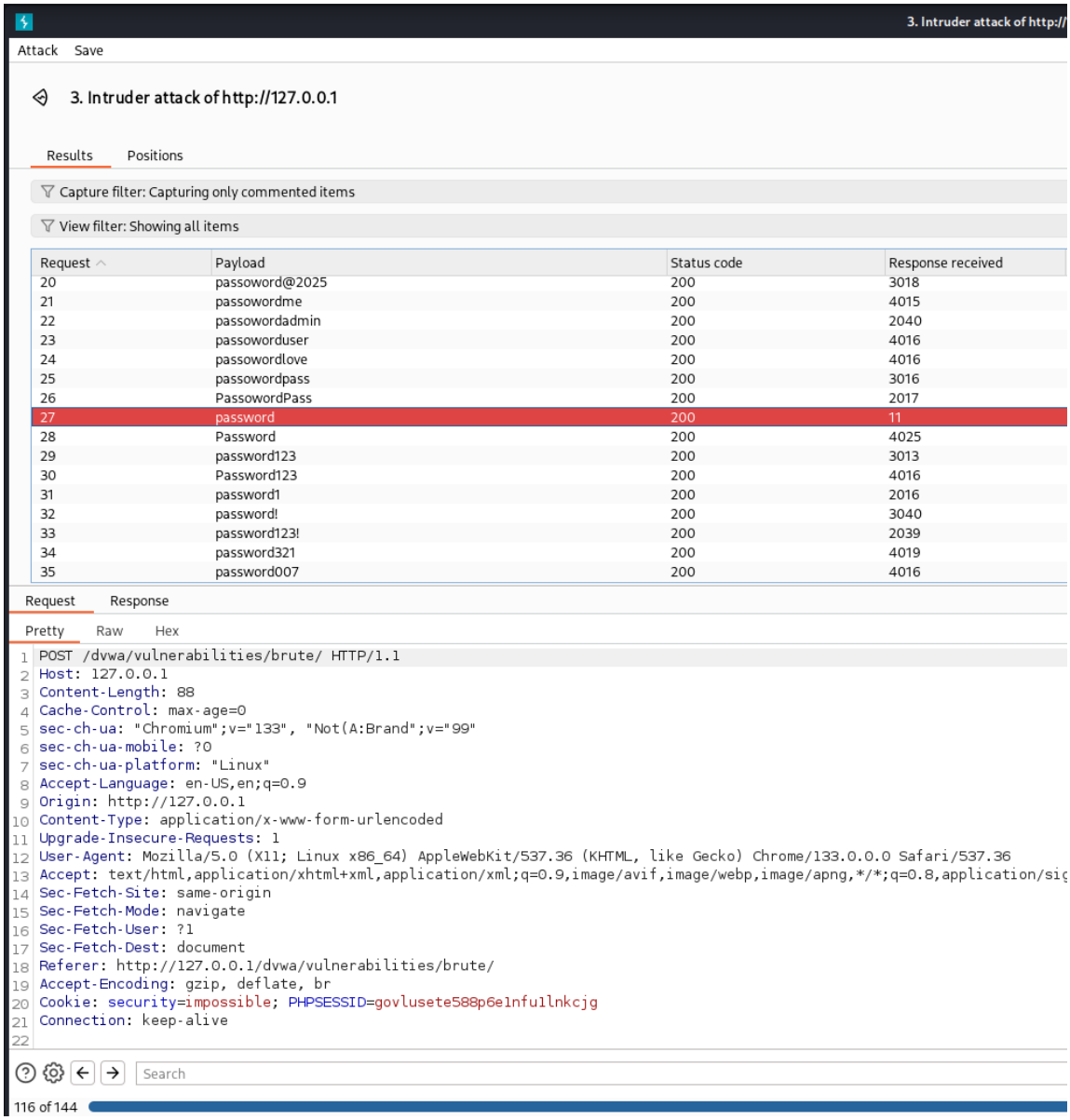


- Loaded a custom wordlist of potential passwords



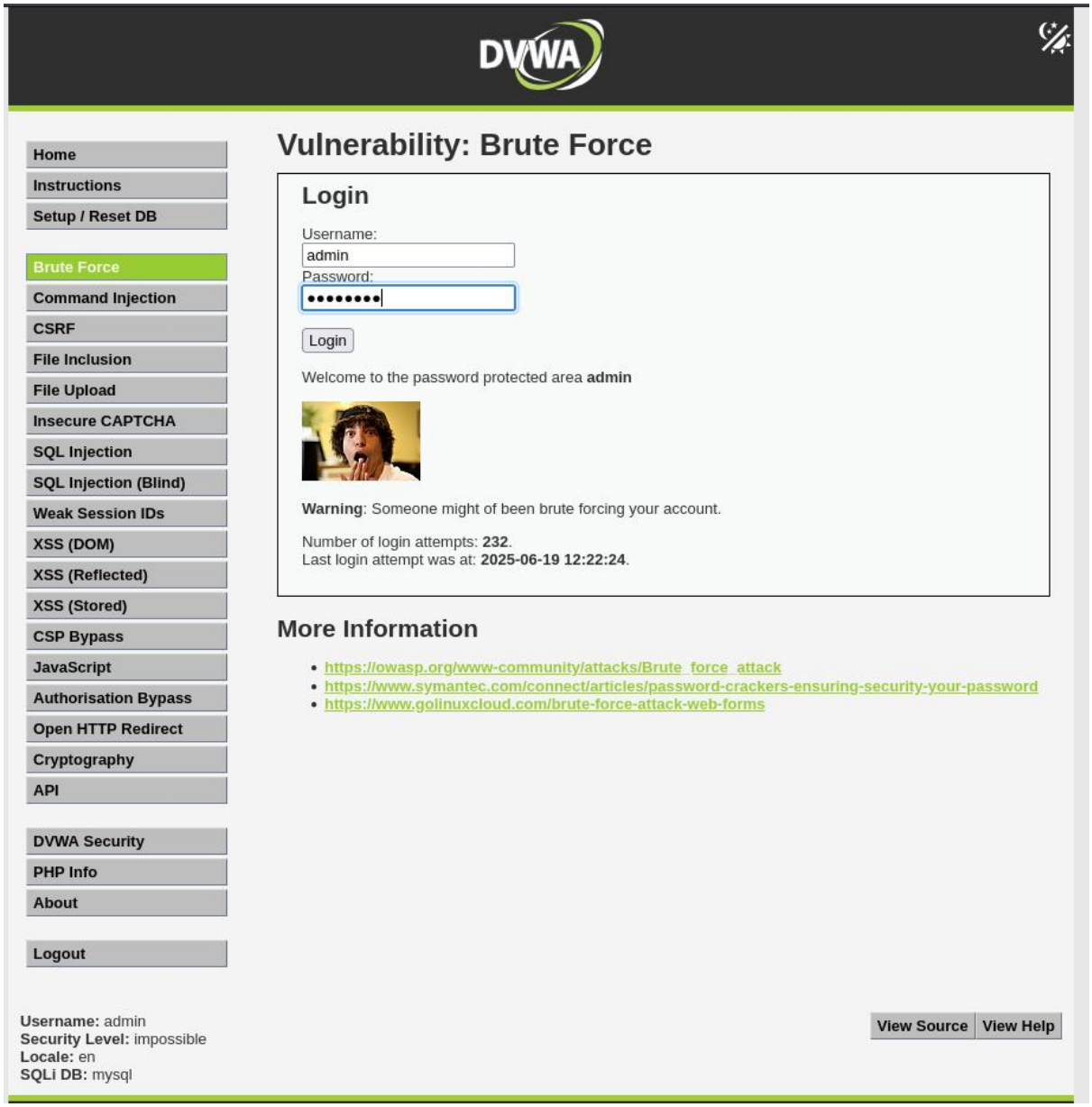
5. Execute Attack:

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



Result:

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



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 defence mechanisms.

✅ **Overall Conclusion**

The security testing conducted on DVWA successfully demonstrated how common vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and weak authentication controls can be exploited to compromise a web application. Each attack revealed critical flaws that could lead to data breaches, unauthorized access, and severe reputational and financial damage in real-world scenarios. This assessment underlines the need for:

- Proper input validation and output encoding
- Use of secure coding practices (e.g., prepared statements)
- Robust authentication mechanisms with rate limiting, CAPTCHA, and multi-factor authentication
- Regular security audits and penetration testing