Vulnerability Assessment Report

Mohammed Firdaws Alnuur

Future Interns

Nairobi, Kenya

Feb 2025

Declaration and approval

I declare that this report is my original work and has not been previously submitted for approval. To the best of my knowledge, all sources have been cited appropriately

Student Name: Mohammed Firdaws Alnuur

Email: firdawsalnuur4@gmail.com

Abstract

This report documents the vulnerability assessment of a sample web application using tools such as OWASP ZAP and Nmap**.** The assessment aimed to identify security vulnerabilities, evaluate their potential impact, and provide actionable recommendations to mitigate risks. The findings highlight common vulnerabilities such as SQL Injection**,** Cross-Site Scripting (XSS)**,** and Insecure Authentication Mechanisms**.** The report concludes with recommendations for improving the security posture of the web application, including code fixes, configuration changes, and the adoption of secure coding practices.

Chapter 1: Introduction

1.1 Background Information

In the modern digital era, web applications are increasingly targeted by cybercriminals due to their widespread use and potential vulnerabilities. Vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Insecure Authentication can lead to data breaches, unauthorized access, and financial losses. This report focuses on identifying such vulnerabilities in a sample web application and providing recommendations to mitigate them.

For this assessment, the Damn Vulnerable Web Application (DVWA) was used as the target application. DVWA is a PHP/MySQL-based web application specifically designed to be vulnerable for educational and testing purposes. Its primary goal is to aid security professionals, developers, and students in understanding and practicing web application security in a controlled environment. DVWA includes various vulnerabilities, such as SQL Injection, XSS, and insecure authentication mechanisms, making it an ideal platform for conducting vulnerability assessments and penetration testing.

1.2 Problem Statement

Web applications often contain vulnerabilities that can be exploited by attackers. These vulnerabilities may arise from insecure coding practices, misconfigurations, or a lack of proper security testing. The Damn Vulnerable Web Application (DVWA) was chosen for this assessment because it intentionally includes common vulnerabilities, allowing for a comprehensive evaluation of security weaknesses.

The goal of this assessment is to identify these vulnerabilities, assess their impact, and provide recommendations to enhance the security of the web application. By using DVWA, this report aims to demonstrate how vulnerabilities can be exploited in real-world scenarios and how they can be mitigated through secure coding practices, configuration changes, and the implementation of security best practices.

Chapter 2. Methodology

Introduction

The assessment was performed using a combination of automated scanning and manual testing. The methodology followed:

2.1 Development approach

The project followed an iterative development approach, allowing for continuous testing and refinement. The key steps included:

1. Reconnaissance & Mapping – Using Nmap A network scanning tool used to identify open ports, services, and potential vulnerabilities in the web application's infrastructure.
2. Scanning & Analysis

- Used OWASP ZAP to perform an automated scan of the web application.
- Identified common vulnerabilities such as SQL Injection, XSS, and insecure headers.

3. Exploitation (Ethical Testing) - Verifying detected vulnerabilities.
4. Reporting - Documenting findings and recommendations.

Findings and Analysis

Analysed the results from both automated and manual testing then Prioritized vulnerabilities based on their severity and potential impact.

ID	Vulnerability Name	Severity	Description
VULN-01	SQL Injection	High	Input fields allow direct interaction with database queries.

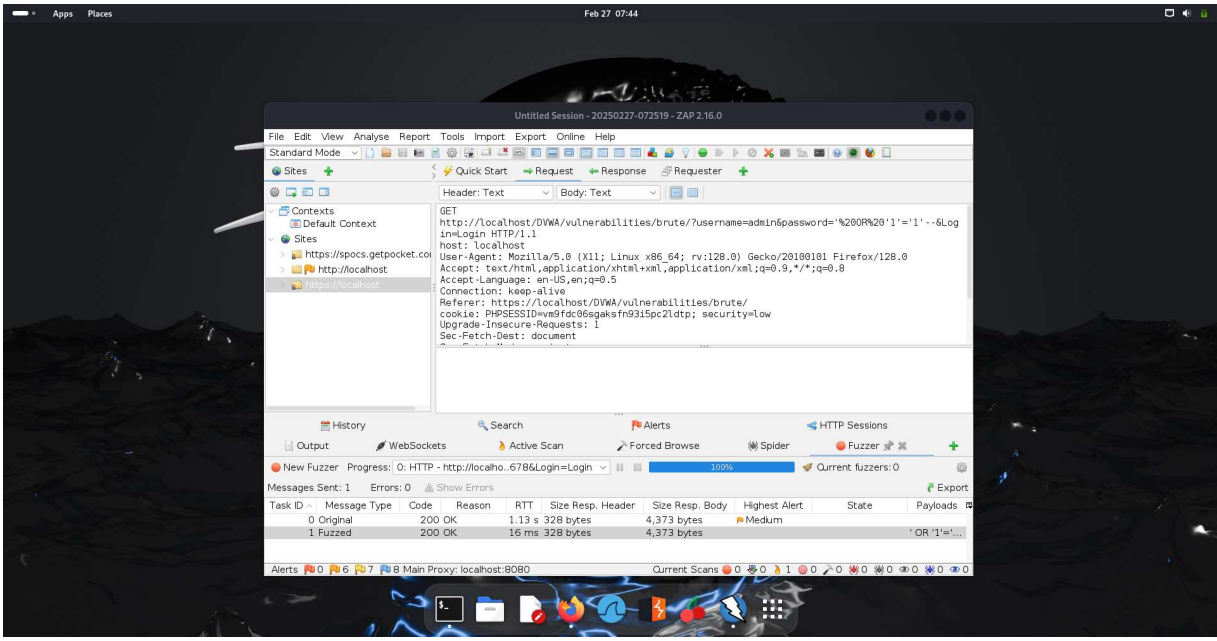
VULN-02	Cross-Site Scripting (XSS)	Medium	User input is not properly sanitized, allowing script injection.
VULN-03	Cookie No Http Only Flag	Medium	Cookies are not marked with the Http Only flag, making them accessible to JavaScript and increasing the risk of session hijacking.
VULN-04	Server Misconfiguration	Medium	Web server exposes sensitive information.
VULN-05	Cookie without Same Site Attribute	Medium	Cookies do not have the Same Site attribute, which could allow Cross-Site Request Forgery (CSRF) attacks.
VULN-06	Server Leaks Version Information via "Server" HTTP Header	Low	The server leaks version information in the HTTP response header, which could provide attackers with valuable information for targeted attacks.

2.4 Detailed Findings

2.4.1 SQL Injection (High Severity)

Description: User input fields do not properly sanitize SQL queries, allowing attackers to manipulate database queries.

- **Tools Used:** An open-source web application security scanner used to identify vulnerabilities such as SQL Injection, XSS, and insecure configurations.
- **Proof of Concept:**
    - Injected payload: ' OR 1=1 --
    - Successful authentication bypass detected.



- **Impact:**
    - Unauthorized data access
- **Recommendation:** Use parameterized queries or prepared statements to prevent SQL Injection attacks.

2.4.2 Cross-Site Scripting (XSS) (Medium Severity)

- **Description:** The application fails to properly encode user-generated input, allowing JavaScript injection.
- **Tools Used:** OWASP ZAP
- **Proof of Concept:**

  <script>alert('XSS')</script> executed successfully.

- **Impact:** Attackers can steal user sessions, deface the website, or redirect users to malicious sites.
- **Recommendation:** Implement input validation and output encoding to prevent XSS attacks.

2.4.3 Insecure Authentication: (High Severity)

- **Description:** The application uses weak password policies and does not enforce multi-factor authentication (MFA).
- **Tools Used:** OSWAP ZAP
- **Impact:** Data leakage and Unauthorized account access
- **Recommendation:** Implement proper access controls and validate user permissions before allowing access to resources.

2.4.4 Server Misconfiguration (Medium Severity)

- **Description:** The server leaks sensitive information through HTTP headers and misconfigured services.
- **Tools Used:** Nmap
- **Impact:** Attackers gain insights into server technology.
- **Recommendation:** Remove or obfuscate server version information in HTTP headers.

2.4.5 Weak Authentication (High Severity)

- **Description:** The application lacks multi-factor authentication (MFA) and enforces weak password policies.
- **Tools Used:** Manual Testing, OWASP ZAP
- **Impact:**
    - Increased risk of brute-force attacks.
- **Recommendation:** Enforce strong password policies and implement multi-factor authentication (MFA).

2.4.6 Content Security Policy (CSP) Header Not Set:

Description: The absence of a CSP header leaves the application vulnerable to XSS and data injection attacks. A CSP header would restrict the sources from which content can be loaded.

Recommendation: Implement a CSP header to restrict the sources from which content can be loaded.

2.4.7 Missing Anti-Clickjacking Header:

Description: The absence of the X-Frame-Options header makes the application vulnerable to clickjacking attacks, where an attacker can trick users into clicking on hidden elements.

Recommendation: Add the X-Frame-Options header to prevent clickjacking attacks.

2.4.8 X-Content-Type-Options Header Missing:

Description: The absence of this header could allow browsers to interpret files as a different MIME type, potentially leading to security risks such as script execution.

Recommendation: Add the X-Content-Type-Options header with the value nosniff to prevent MIME type sniffing.

2.4.9 Cookie No HttpOnly Flag:

Description: Cookies without the HttpOnly flag are accessible to JavaScript, increasing the risk of session hijacking through XSS attacks.

Recommendation: Mark cookies with the HttpOnly flag to prevent access via JavaScript.

2.4.10 Cookie without SameSite Attribute:

Description: Cookies without the SameSite attribute are vulnerable to CSRF attacks, where an attacker can force a user to perform unwanted actions.

Recommendation: Add the SameSite attribute to cookies to prevent CSRF attacks.

2.4.11 Server Leaks Version Information:

Description: The server leaks version information in the HTTP response header, which could provide attackers with valuable information for targeted attacks.

Recommendation: Remove or obfuscate server version information in HTTP headers.

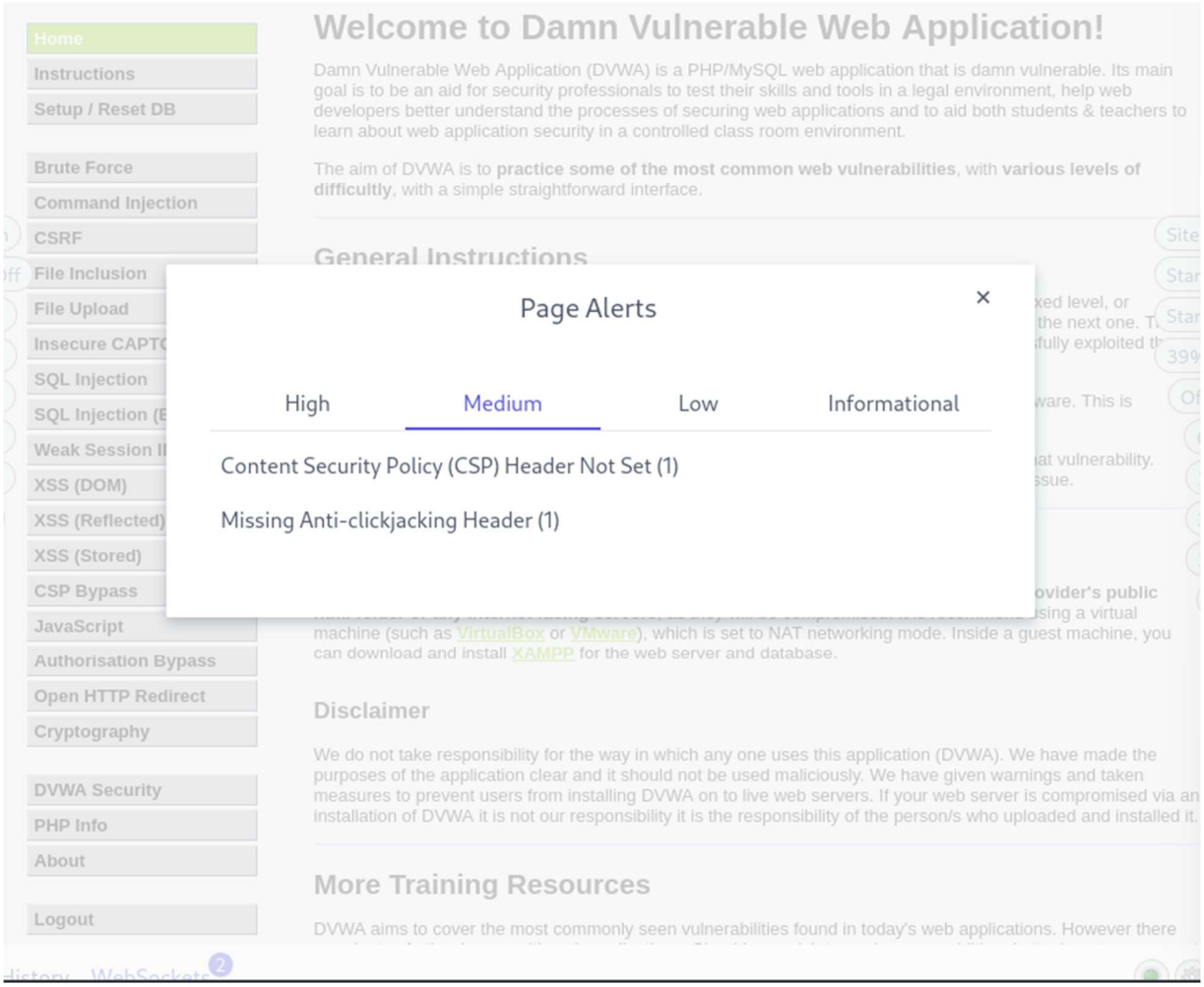
2.4.12 Conclusion and Recommendations

The assessment identified multiple security vulnerabilities that could be exploited to compromise the web application. Key recommendations include:

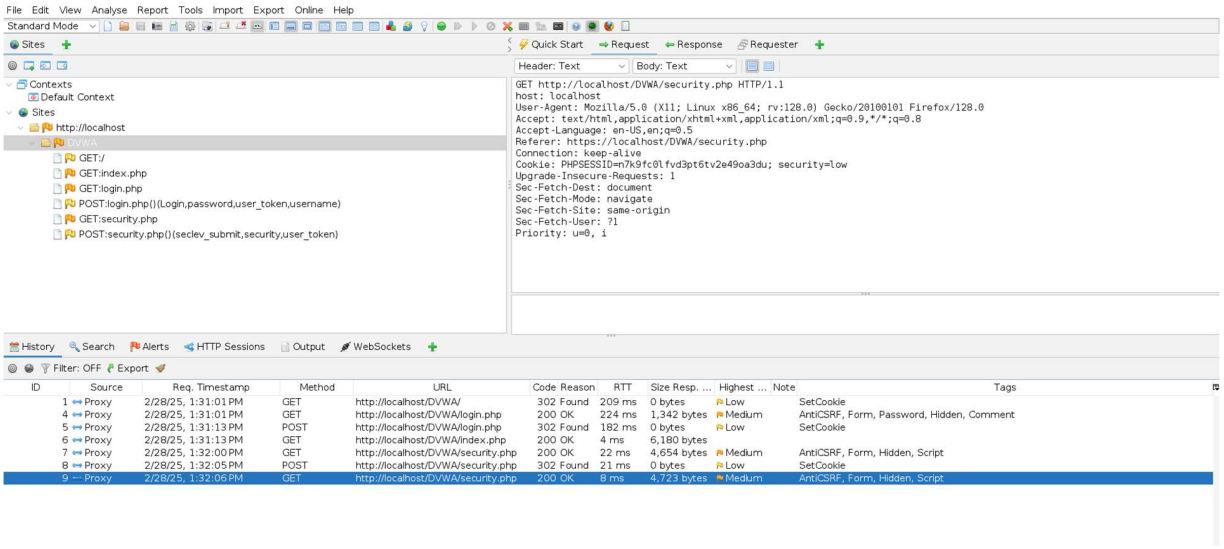
1. **Sanitize and validate all user inputs** to prevent SQL injection and XSS.
2. **Enforce strong authentication mechanisms** including multi-factor authentication.
3. **Implement strict access controls** to prevent unauthorized access to resources.
4. **Secure server configurations** by disabling unnecessary services and removing sensitive information from responses.
5. **Regular security testing and patching** to mitigate future vulnerabilities.

Chapter 3. Appendices

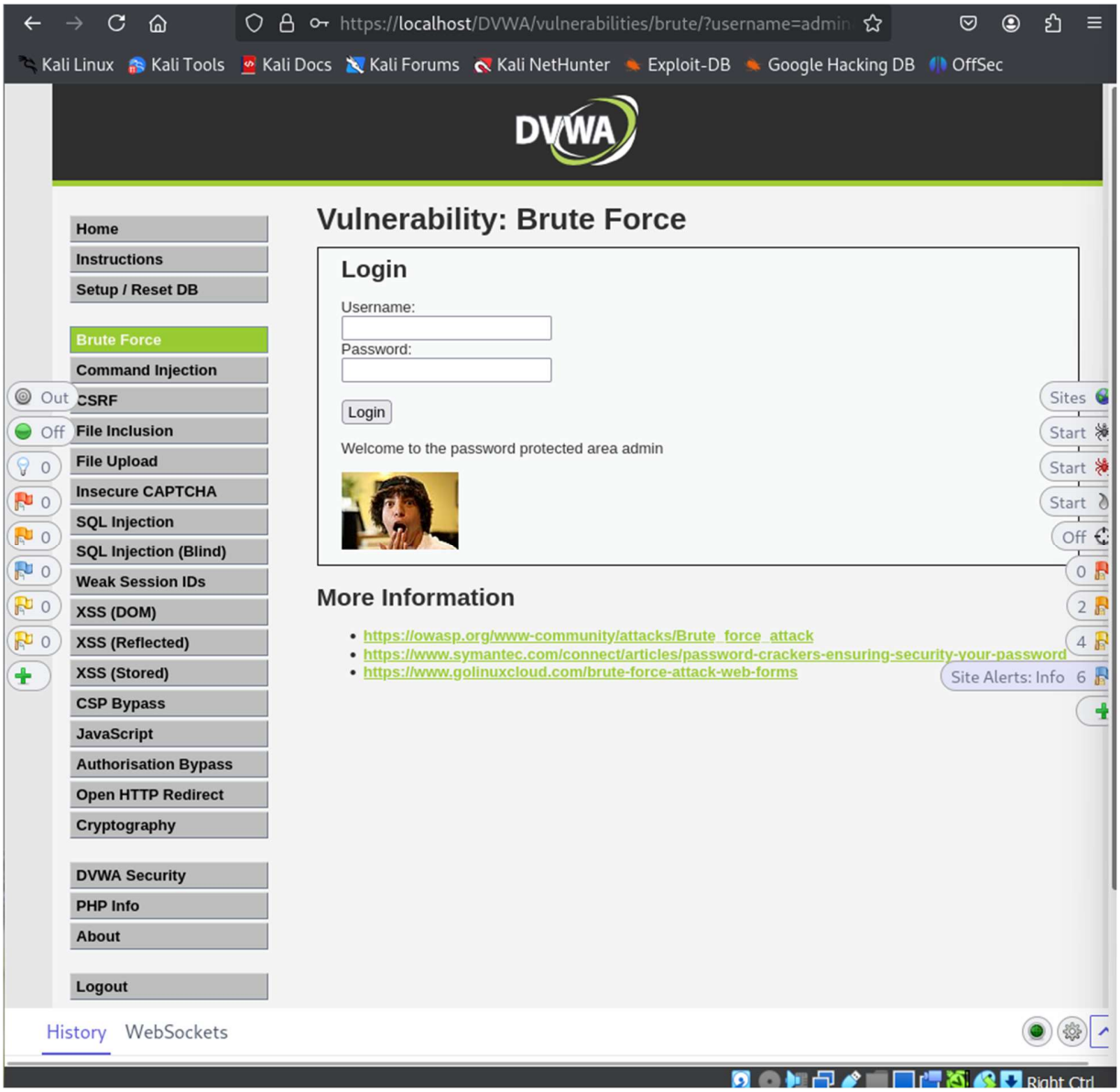
1. **Alerts.png** – Displays general security alerts detected during the vulnerability assessment.



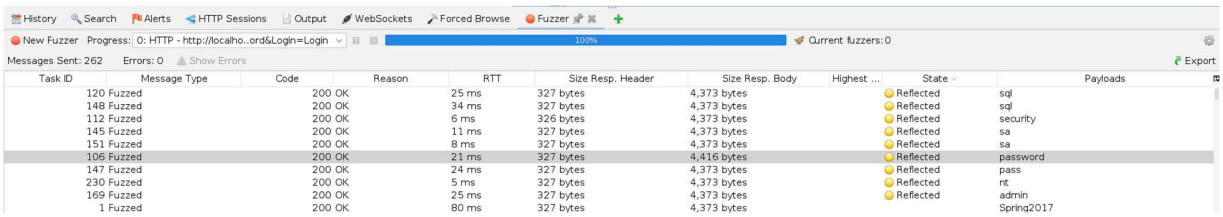
2. **Authenticated Session.png** – Shows a logged-in session, which may indicate session handling or authentication issues.



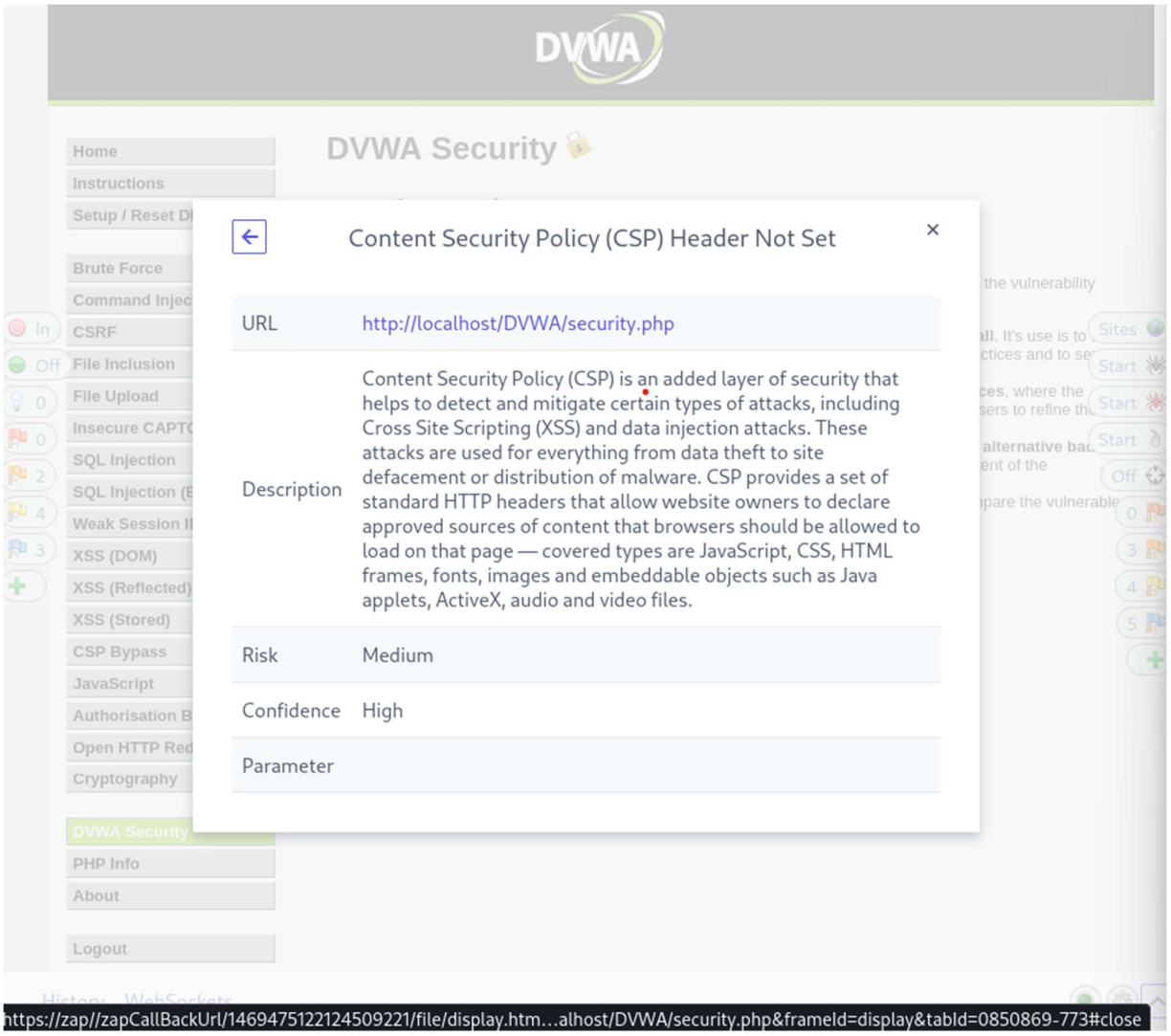
3. **Brute Force.png** – Screenshot of a brute force attack attempt on the login system.



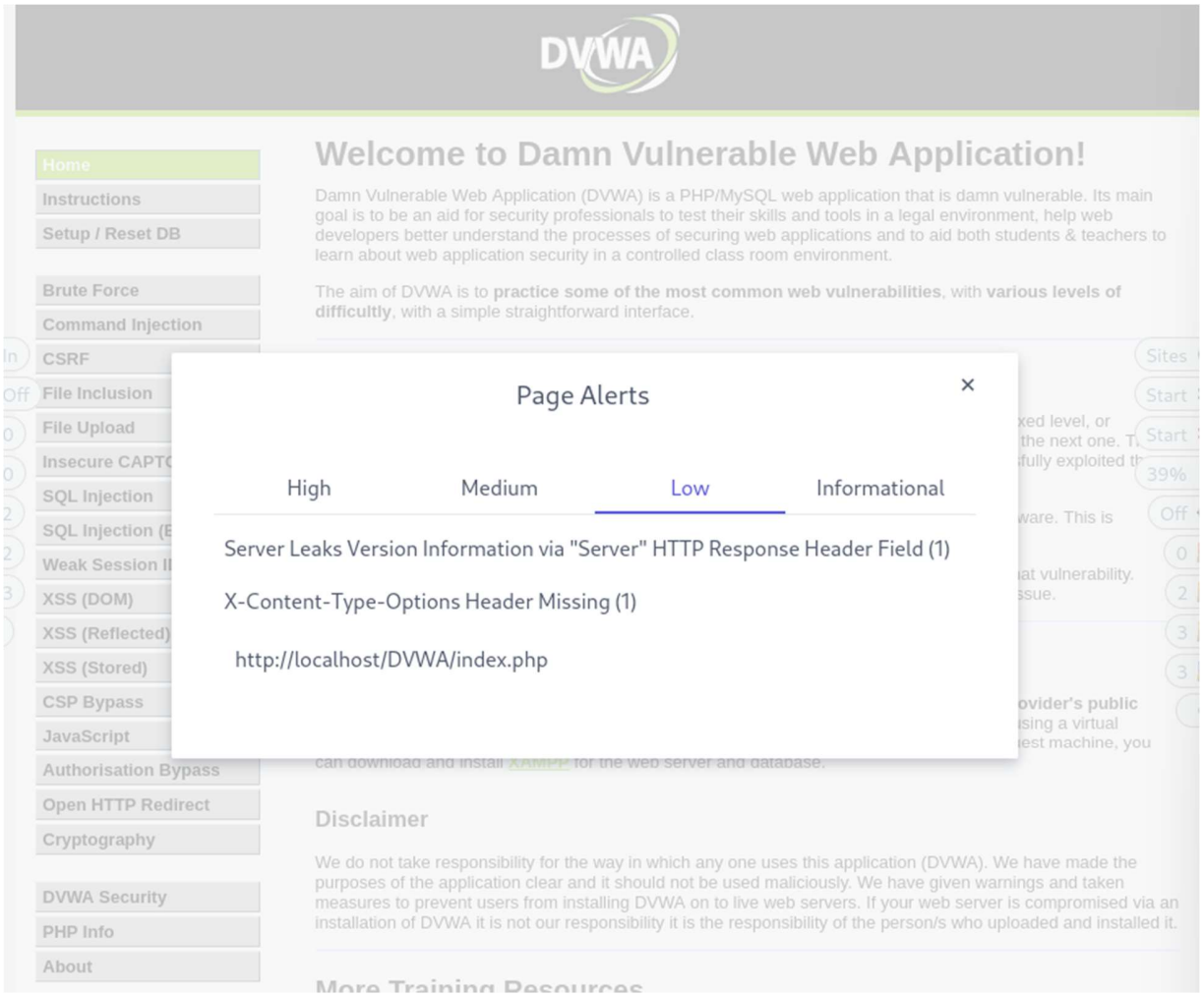
4. **Brute Force Attack Success.png** – Evidence of a successful brute force attack, demonstrating weak authentication security.



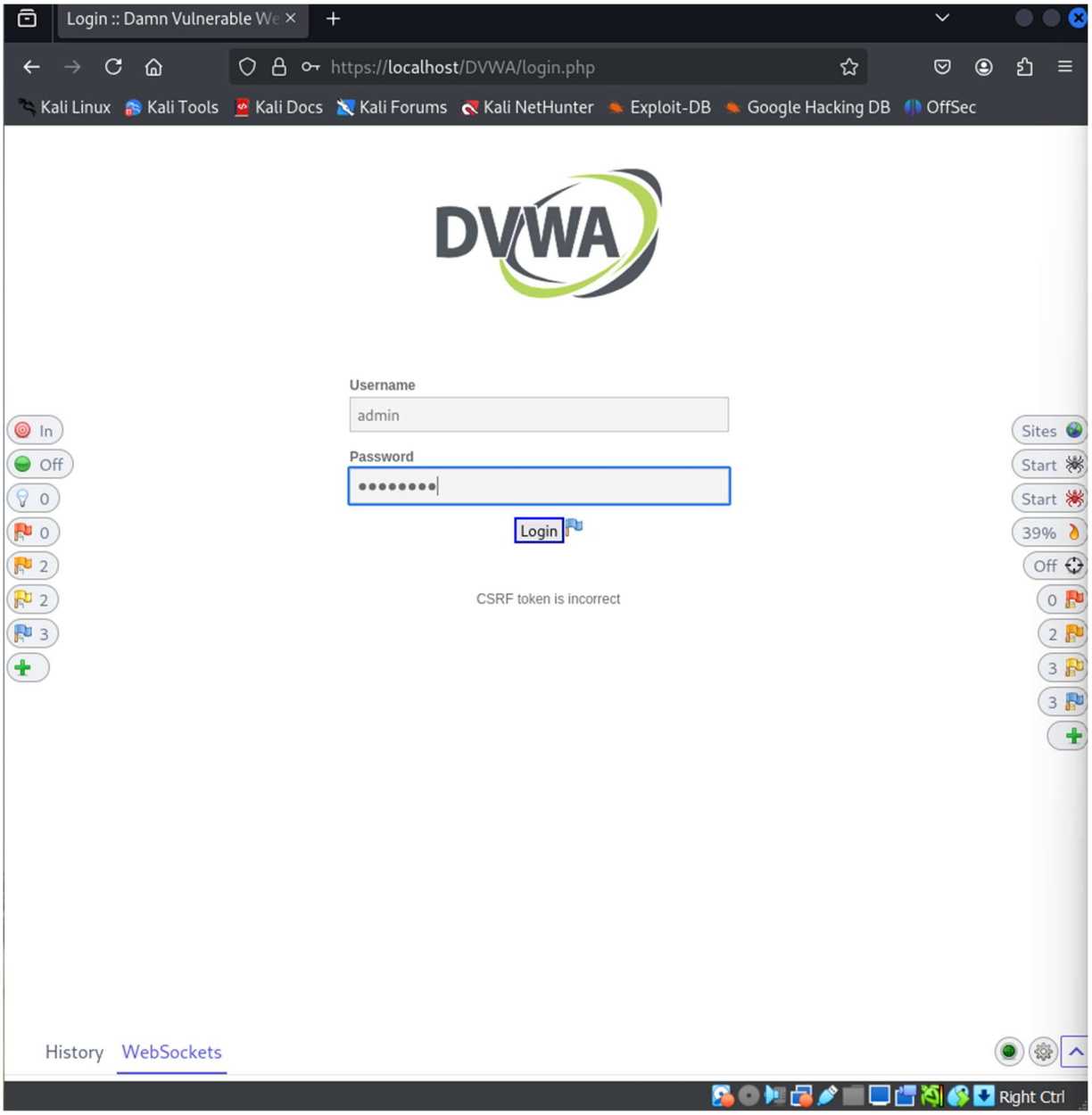
5. **Content Security Policy (CSP) Header Not Set** – Indicates a missing CSP header, which could allow XSS and other injection attacks.



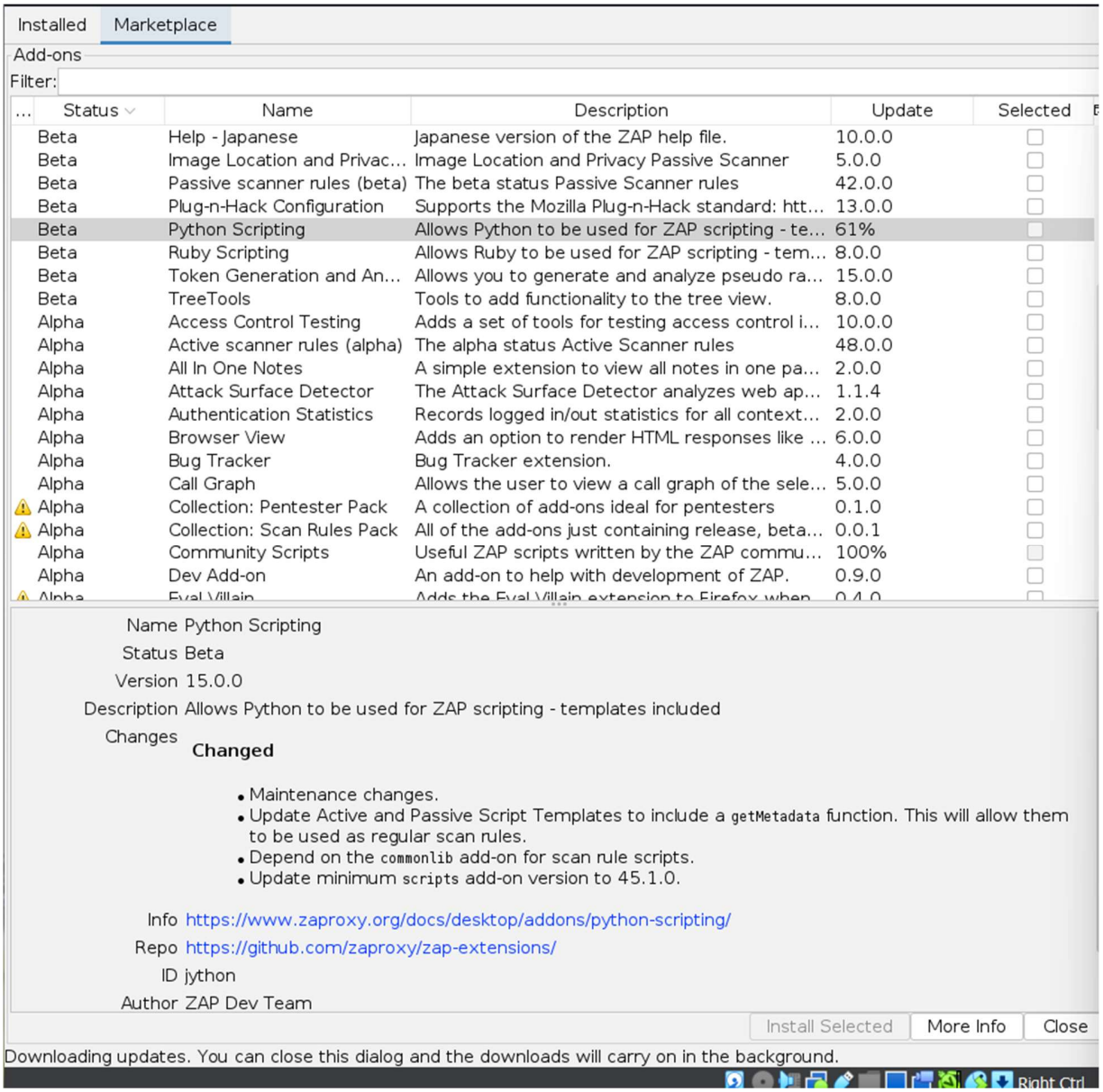
6. **X-Content-Type-Options Header Missing (1) Alert.png** – Indicates the absence of the `X-Content-Type-Options` header, making the application vulnerable to MIME-based attacks.



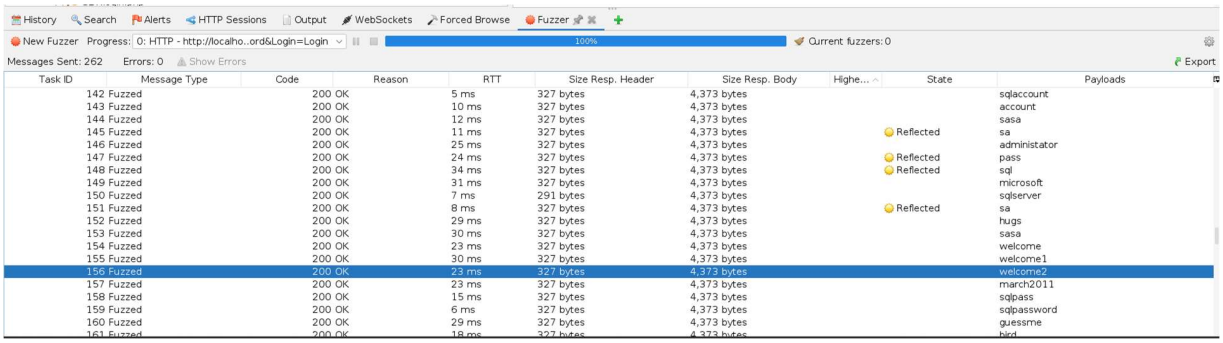
7. **LOGIN ACTIVE Scan.png** – Displays an active vulnerability scan on the login page.



8. **Manage Add-Ons.png** – Shows browser security configurations and add-ons that could affect the assessment.



9. **Successfully Fuzzed Brute Force Attack.png** – Demonstrates a successful fuzzing attack that bypassed authentication.



Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highe...	State	Payloads
142	Fuzzed	200	OK	5 ms	327 bytes	4,373 bytes			sqlaccount
143	Fuzzed	200	OK	10 ms	327 bytes	4,373 bytes			account
144	Fuzzed	200	OK	12 ms	327 bytes	4,373 bytes			sasa
145	Fuzzed	200	OK	11 ms	327 bytes	4,373 bytes		Reflected	sa
146	Fuzzed	200	OK	25 ms	327 bytes	4,373 bytes			administator
147	Fuzzed	200	OK	24 ms	327 bytes	4,373 bytes		Reflected	pass
148	Fuzzed	200	OK	34 ms	327 bytes	4,373 bytes		Reflected	sql
149	Fuzzed	200	OK	31 ms	327 bytes	4,373 bytes			microsoft
150	Fuzzed	200	OK	7 ms	291 bytes	4,373 bytes			sqlserver
151	Fuzzed	200	OK	8 ms	327 bytes	4,373 bytes		Reflected	sa
152	Fuzzed	200	OK	29 ms	327 bytes	4,373 bytes			hugs
153	Fuzzed	200	OK	30 ms	327 bytes	4,373 bytes			sasa
154	Fuzzed	200	OK	23 ms	327 bytes	4,373 bytes			welcome
155	Fuzzed	200	OK	30 ms	327 bytes	4,373 bytes			welcome1
156	Fuzzed	200	OK	23 ms	327 bytes	4,373 bytes			welcome2
157	Fuzzed	200	OK	23 ms	327 bytes	4,373 bytes			march2011
158	Fuzzed	200	OK	15 ms	327 bytes	4,373 bytes			sqlpass
159	Fuzzed	200	OK	6 ms	327 bytes	4,373 bytes			sqlpassword
160	Fuzzed	200	OK	29 ms	327 bytes	4,373 bytes			guessme
161	Fuzzed	200	OK	18 ms	327 bytes	4,373 bytes			bird