**DVWA Security Assessment Report**

**1. Introduction**

This report presents the findings of a security assessment performed on the **Damn Vulnerable Web Application (DVWA)**, a deliberately vulnerable PHP/MySQL web app used for penetration testing practice and web security education.

The objective was to simulate real-world attacks on common web vulnerabilities and understand how insecure coding practices can be exploited. The test was conducted in a controlled lab setup using industry-standard tools like **Burp Suite**, **SQLMap**, and **FoxyProxy**.

This engagement focused on three key areas from the OWASP Top 10 vulnerabilities:

* **SQL Injection**
* **Reflected Cross-Site Scripting (XSS)**
* **Broken Authentication**

**2. Target Environment**

* **Application**: Damn Vulnerable Web Application (DVWA)
* **Platform**: Windows 10
* **Web Stack**: XAMPP (Apache, MySQL, PHP)
* **Browser**: Burp Suite Browser (Chromium)
* **Tools Used**:
  + Burp Suite (for traffic interception & manipulation)
  + SQLMap (for automating SQLi attacks)
  + Python (optional for scripting)
* **DVWA Security Level**: Set to “Low”
* **Test URL**: <http://localhost/dvwa/>

**3. Identified Vulnerabilities and Mitigation Strategies**

**3.1 SQL Injection (SQLi)**

**Description**:  
The SQL Injection module of DVWA allows unvalidated user input to be concatenated directly into SQL queries. Attackers can manipulate this input to alter query logic and gain unauthorized access.

**Demonstration**:  
Input used in the user ID field: id=1' OR '1'='1

This input returned all users from the database, proving that the backend query was modified.

**Automated Exploitation**:

SQLMap was able to enumerate the database names, tables, and user credentials.

**Impact**:

* Data disclosure
* Database modification
* Authentication bypass
* Full database compromise

**Mitigation**:

* Use **parameterized queries** or **prepared statements**
* Implement **input validation/sanitization** on all user inputs
* Apply **least privilege** principles for database users
* Deploy **Web Application Firewalls (WAFs)**

**3.2 Reflected Cross-Site Scripting (XSS)**

**Description**:  
DVWA's Reflected XSS module accepts unsanitized user input and reflects it back into the HTML response. Attackers can inject JavaScript that executes in the browser context.

**Demonstration**:  
Input: <script>alert('XSS')</script>

Upon submission, an alert box appeared in the browser, confirming execution of the injected script.

**Impact**:

* Session hijacking
* Cookie theft
* Redirecting users to malicious sites
* Phishing attacks

**Mitigation**:

* **Encode output** based on context (HTML, JavaScript, attributes, etc.)
* Apply **input sanitization** both client- and server-side
* Implement **Content Security Policy (CSP)** headers
* Use libraries like **DOMPurify** to clean DOM elements

**3.3 Broken Authentication (Weak Credentials)**

**Description**:  
DVWA’s Brute Force module demonstrates poor authentication mechanisms. Login was successful using the default credentials:

Username: admin

Password: password

The system had no account lockout, no CAPTCHA, and allowed unlimited attempts.

**Impact**:

* Unauthorized access to admin interface
* Privilege escalation
* Sensitive data exposure

**Mitigation**:

* Enforce **strong password policies**
* Remove or disable **default credentials**
* Implement **login rate-limiting** and **account lockout** mechanisms
* Use **multi-factor authentication (MFA)** where possible

**4. Conclusion**

This assessment revealed several critical and common vulnerabilities in DVWA:

* **SQL Injection** was successfully exploited to extract database content.
* **Reflected XSS** allowed client-side JavaScript execution.
* **Broken Authentication** enabled access using default credentials.

Although DVWA is intentionally insecure for training purposes, the test highlights how easily unprotected systems can be compromised if secure coding principles are ignored.

**Key Takeaways:**

* Always validate and sanitize user input
* Never trust client-side data
* Use secure coding patterns (e.g., parameterized queries)
* Implement layered security controls (WAFs, rate limiting, CSP)

**Prepared by**: *NNAJI DANIEL CHINWEOKE*  
**Internship**: *FutureInterns – Cybersecurity Internship*  
**Date**: *07 August 2025*