**Web Application Security Assessment and SQL Injection Testing Report**

**Introduction**

This report outlines the findings of a web application security assessment performed on **Damn Vulnerable Web Application (DVWA)** to evaluate its susceptibility to SQL injection attacks. The goal was to understand the functionality of the application, identify SQL injection vulnerabilities, and document exploitation methods while emphasizing the importance of secure coding practices.

**Initial Assessment**

**Application Features:**

* Login functionality.
* Input fields for searching or querying data.
* Forms requiring user input.

**Entry Points Identified:**

* **Login Form:** The username and password fields.
* **ID Parameter:** Used in URL queries.
* **Search Box:** Accepts user-defined input.

**SQL Injection Testing**

**1. Basic SQL Injection**

**Objective:** Test the application's response to simple SQL injection payloads.

**Steps Performed:**

* Input: ' OR '1'='1 in login forms and input fields.

**Observation:** Successful login or data retrieval without valid credentials.

**2. Advanced SQL Injection**

**Blind SQL Injection**

* **Payload:** 1' AND 1=1 -- (Valid) and 1' AND 1=2 -- (Invalid).
* **Observation:** Differing responses indicated potential vulnerabilities.

**Time-Based Blind Injection**

* **Payload:** 1' AND SLEEP(5) --.
* **Observation:** Noticeable delays in server response.

**3. Automated Testing with SQLMap**

**Command:**

sqlmap -u "http://<DVWA-URL>?id=1" --dbs

**Results:**

* Extracted database names and table structures.

**Hands-On Exploitation**

**Exploitation Steps**

1. **Extracting User Data:**
   * **SQL Payload:** 1' UNION SELECT username, password FROM users --.
   * **Outcome:** Retrieved sensitive user data.
2. **Database Schema Extraction:**
   * Used UNION-based SQL injection to enumerate tables and columns.

**Risks Identified**

* Data leakage.
* Unauthorized database access.
* Potential for application-level privilege escalation.

**Recommendations**

To mitigate SQL injection vulnerabilities, implement the following secure coding practices:

1. **Input Validation:**
   * Validate all user inputs to ensure they conform to expected formats.
2. **Use of Prepared Statements:**
   * Implement parameterized queries to prevent SQL code execution.
3. **Error Handling:**
   * Avoid displaying detailed database error messages.
4. **Regular Security Testing:**
   * Conduct frequent vulnerability assessments using tools and manual techniques.
5. **Secure Authentication Mechanisms:**
   * Enforce strong passwords and implement multi-factor authentication.

**Conclusion**

This assessment demonstrated the risks posed by SQL injection vulnerabilities and highlighted the importance of secure coding practices. Exploitation of these vulnerabilities can lead to severe consequences, such as unauthorized access, data leakage, and system compromise. By adopting robust input validation, parameterized queries, and regular security assessments, organizations can effectively mitigate such threats and strengthen their web application security.