API Security Testing Lab Report

1. Introduction

The purpose of this lab was to perform **API Security Testing** on a vulnerable web application (DVWA environment) using **Burp Suite, Postman, and sqlmap**. The testing objective was to evaluate the API for weaknesses defined in **OWASP API Security Top 10**, identify exploitability, analyze impact and document findings with professional methodology.

The environment simulated a real-world scenario where insecure APIs expose sensitive user data, authentication flaws and injection points.

2. Lab Environment

Component	Details
Target Application	DVWA (Damn Vulnerable Web Application)
Mode	Security Level – Low
Tools Used	Burp Suite, Postman, sqlmap, browser proxy
Testing Approach	Manual + Automated + Active Testing
Authentication	Default admin login for access
Test Machine	Kali Linux / Windows attacker
Target URL	http://192.168.0.9/

3. Tools & Purpose

Tool	Purpose in API Testing
Burp Suite	Intercept requests, modify parameters, token tampering, replay requests
Postman	Craft API requests, test endpoints manually, brute & parameter fuzz
sqlmap	Automated SQL injection testing on parameters/endpoints
Browser	UI interaction to generate API calls for interception

4. Methodology

We followed a structured VAPT methodology:

1. **Reconnaissance & Endpoint Enumeration**
2. **Authentication & Token Analysis**
3. **Parameter Tampering**
4. **Injection Testing (SQL)**
5. **Authorization Testing**
6. **Rate Limiting & Access Control**
7. **Reporting & Evidence Logging**

5. API Endpoints Identified & Tested

Endpoints captured using **Burp Suite Proxy + Manual browsing**

Method	URL / Endpoint	Purpose
POST	/login.php	User authentication endpoint
GET/POST	/vulnerabilities/csrf/	Password change request
GET	/vulnerabilities/sqli/?id=	Data fetch parameter-based
GET/POST	/api/users (simulated lab)	User details listing

DVWA does not provide native APIs, so parameter-driven PHP endpoints were treated as API-like request handlers (same exploitation behaviour).

6. Detailed Testing Activities

6.1 Endpoint Enumeration

Objective: Identify API routes & hidden endpoints

Tools: Burp Proxy + Repeater + Postman

Steps:

- Started Burp Suite proxy and visited DVWA pages.
- Captured traffic, exported requests to Postman.
- Verified endpoints manually by modifying parameters.

Findings:

Multiple endpoints accepted client-side parameters without validation — potential for injection, CSRF, BOLA-like attacks.

6.2 Authentication & Token Testing

Tools: Burp Suite Repeater, Postman

Test Performed:

- Captured login request /login.php
- Removed CSRF token & replayed request
- Attempted login with invalid/missing session cookie

Result:

Application **accepted request even with manipulated token** in Low mode → **CSRF insecure implementation**

Impact:

- Attacker could force users to change passwords through malicious links.

6.3 Burp Suite – CSRF Password Change Exploit

Workflow:

1. Navigated to vulnerabilities/csrf/
2. Intercepted the request for **password change**
3. Modified request and forwarded without verification
4. Constructed malicious GET request URL format:

http://192.168.0.9/vulnerabilities/csrf/?password_new=attacker123&password_conf=attacker123&Change=Change

Result:

User password changed **without authentication challenge**

Severity: **High/Critical**

6.4 SQL Injection Testing (Manual + sqlmap)

Target Endpoint:

http://192.168.0.9/vulnerabilities/sqli/?id=1&Submit=Submit

Manual Injection attempts:

?id=1'

?id=1 OR 1=1--

?id=1 UNION SELECT null, user()

Observation:

Application returned valid output → **Injection confirmed**

Automated Scan using sqlmap:

sqlmap -u "http://192.168.0.9/vulnerabilities/sqli/?id=1&Submit=Submit" --dbs --batch

Findings:

- Dumped database schema
- Retrieved users table
- SQLi present due to unsanitized parameters

Severity: **Critical**

6.5 BOLA/API Authorization Testing

Objective: Check if unauthorized sessions can access data.

Steps:

- Captured request from authenticated session
- Removed session cookie
- Replayed via Burp Repeater
- Tested user enumeration endpoint /api/users (simulated lab behaviour)

Result:

❗ Without session token, application still served response (Low config)

→ **Broken Object Level Authorization**

Impact:

- Any unauthenticated user could extract user information.

Severity: **Critical**

6.6 Postman Manual API Manipulation

Activities Done:

Test	Observation
Changed parameters	App executed without server-side validation
Invalid input fuzzing	Work fuzzing
Removed headers/tokens	No rejection under low security

Outcome:

- Validated lack of proper API validation on input & auth layers.

7. Summarized Findings – Professional Sheet

ID	Vulnerability	OWASP API Category	Impact
01	CSRF No Token Validation	API8 – Injection/CSRF	Unauthorized password reset
02	SQL Injection	API8 – Injection	DB extraction & takeover
03	Missing Auth/BOLA	API1 – Broken Object Level Auth	Unauthorized data access

8. Conclusion

The DVWA API testing lab demonstrated critical weaknesses commonly observed in insecure web APIs. Lack of validation, weak authentication, SQL injection and CSRF vulnerabilities allow attackers to hijack accounts, extract databases, and compromise the entire environment. This exercise helped understand:

- How to intercept & alter API requests
- How authentication tokens work & fail

- Manual + automated SQLi exploitation
- Real attack flow using Burp & Postman

9. Recommendations

- Implement server-side CSRF tokens & validation checks.
- Use prepared statements & input sanitization to prevent SQLi.
- Enforce authentication before every sensitive API action.
- Use rate limiting, logging & token expiration.
- Follow OWASP API Security Top 10 compliance guidelines.

API Test Summary

API security testing was conducted on DVWA using Burp Suite, Postman, and sqlmap. Multiple endpoints were enumerated, intercepted, and manipulated to test for OWASP API Top 10 vulnerabilities. Critical issues were identified including SQL Injection, CSRF-based password reset, and broken authorization (BOLA). Token modification, parameter tampering, and automated SQLi exploitation succeeded, allowing unauthorized password change and database extraction. The assessment confirms insecure input handling, weak auth controls, and absence of server-side validation, posing high security risk.