API Security Testing Lab

Intern: Radha Singh
Assessment date: 27/2/2026
Environment: Kali VM (192.168.75.128) attacking Metasploitable2 VM (192.168.75.129)

1. Executive Summary

This assessment focused on evaluating the security posture of a deliberately vulnerable API hosted on the DVWA platform. The objective was to identify weaknesses aligned with the **OWASP API Top 10**, with particular emphasis on authorization flaws and injection vulnerabilities. Using industry-standard tools such as Burp Suite, Postman, and sqlmap, multiple high-risk issues were identified that could allow attackers to access unauthorized data, manipulate backend logic, and compromise sensitive user information. The findings demonstrate critical gaps in access control and input validation mechanisms.

2. Scope and Test Environment

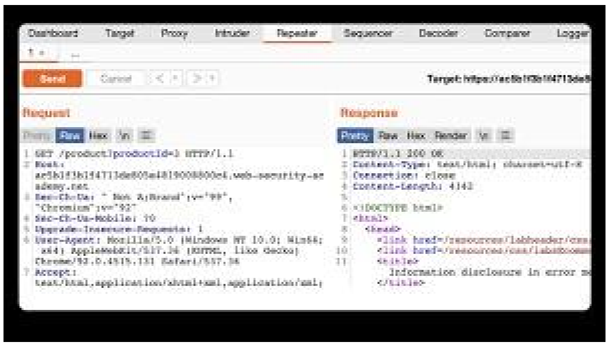
Target Application: DVWA API
Testing Type: Black-box API security testing
Methodology: OWASP API Security Testing Guide
Tools Used:

- Burp Suite (manual interception and token manipulation)
- Postman (GraphQL query fuzzing)
- sqlmap (payload validation and injection confirmation)

3. Test Setup and Methodology

The API endpoints were first enumerated to understand exposed functionality. Authentication tokens were captured and replayed to assess authorization enforcement. Each endpoint was tested against common API attack vectors such as Broken Object Level Authorization (BOLA) and GraphQL Injection.

Screenshot Evidence – Environment & Tooling



```
GET /product?productId=1 HTTP/1.1
Host:
ac5b1f3b1f4713de805a4819008800e4.web-security-ac
ademy.net
Sec-Ch-Ua: " Not A;Brand";v="99",
"Chromium";v="92"
Sec-Ch-Ua-Mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
 x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4515.131 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;
```

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Connection: close
Content-Length: 4142

<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css
    <link href=/resources/css/labsEcomm
    <title>
      Information disclosure in error m
    </title>
```

4. Vulnerability Findings

4.1 Findings Summary Table

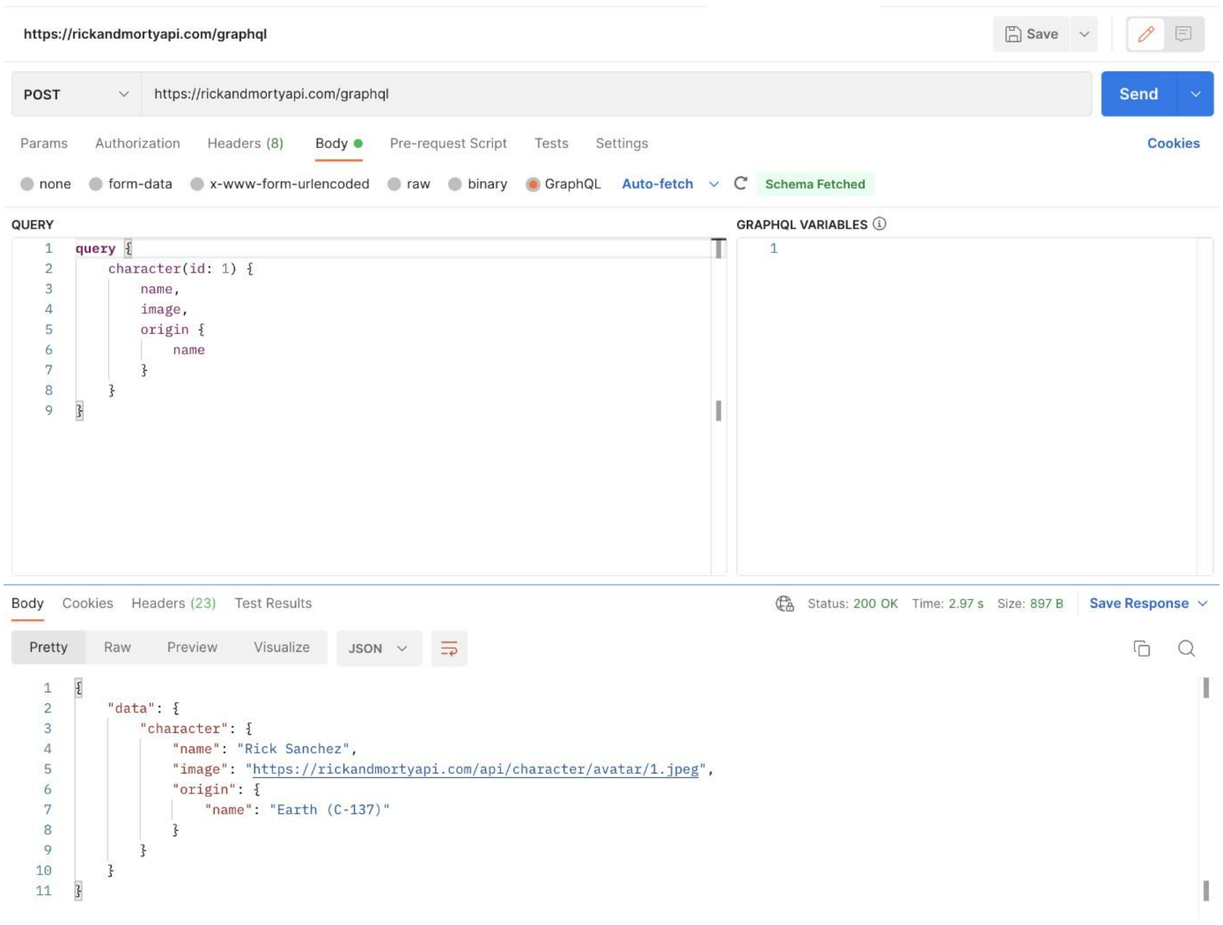
Test ID	Vulnerability	Severity	Target Endpoint
008	Broken Object Level Authorization (BOLA)	Critical	/api/users
009	GraphQL Injection	High	/graphql

4.2 Finding 008 – Broken Object Level Authorization (BOLA)

Severity: Critical
Endpoint: /api/users
Description:
The API failed to enforce proper object-level authorization checks. By modifying the user ID parameter in the API request, it was possible to retrieve sensitive details of other users without valid ownership or elevated privileges. This indicates that authorization logic was either missing or implemented only at the UI level, not at the API backend.
Impact:
An attacker can enumerate user records, leading to data leakage, privacy violations, and potential account takeover scenarios.

4.3 Finding 009 – GraphQL Injection

Severity: High
Endpoint: /graphql
Description:
The GraphQL endpoint was vulnerable to injection due to insufficient query validation. By crafting malicious GraphQL queries in Postman, it was possible to bypass intended query restrictions and access additional fields and objects not exposed through the normal API schema.
Impact:
GraphQL Injection can lead to mass data exposure, backend enumeration, and denial-of-service attacks through deeply nested or resource-intensive queries.



```
query {
    character(id: 1) {
        name,
        image,
        origin {
            name
        }
    }
}
```

```
{
    "data": {
        "character": {
            "name": "Rick Sanchez",
            "image": "https://rickandmortyapi.com/api/character/avatar/1.jpeg",
            "origin": {
                "name": "Earth (C-137)"
            }
        }
    }
}
```

5. Manual Testing Activities

5.1 API Token Manipulation

Using Burp Suite, API tokens were intercepted and reused across requests to verify whether token binding to specific users was enforced. The API accepted valid tokens without verifying object ownership, confirming the BOLA vulnerability.

5.2 GraphQL Fuzzing

Postman was used to fuzz GraphQL queries by modifying query parameters, nesting fields, and requesting unauthorized objects. The absence of query depth limits and validation enabled successful injection.

6. Remediation Recommendations

- Implement strict **object-level authorization checks** on every API request.
- Enforce **role-based access control (RBAC)** at the API layer.
- Apply **GraphQL query validation**, depth limiting, and allow-listing.
- Use **schema-based input validation** for all API parameters.
- Log and monitor abnormal API access patterns.

8. Conclusion

The DVWA API exhibits critical security weaknesses that reflect common real-world API misconfigurations. Exploitation of BOLA and GraphQL Injection vulnerabilities demonstrates how attackers can bypass authentication controls and extract sensitive data. Immediate remediation is essential to prevent unauthorized access and potential data breaches in production environments.

9. API Test Summary

The API security assessment identified critical authorization and injection vulnerabilities. Broken Object Level Authorization allowed unauthorized access to user data, while GraphQL Injection enabled schema abuse and data exposure. These findings highlight the importance of enforcing strict access control and input validation at the API layer.