SECURE API TESTING AND AUTHORIZATION VALIDATION

Table of Contents

1. Abstract and Scope of Assessment

The following report details a deep-dive security analysis of a RESTful API environment. The scope of this assessment includes the identification of logical flaws, authentication bypasses, and resource management issues. With the proliferation of microservices, the security of individual API endpoints has become the critical perimeter for modern enterprises.

The primary target for this assessment is a local instance of the OWASP Juice Shop, an environment specifically engineered to exhibit the most common and dangerous security flaws found in modern web applications. The testing methodology strictly follows the OWASP API Security Project guidelines.

2. Modern API Security Landscape

Traditional web security focused heavily on Cross-Site Scripting (XSS) and SQL Injection at the UI layer. However, APIs require a different perspective. Because APIs expose the underlying business logic, attackers often look for "Mass Assignment" or "IDOR" (Insecure Direct Object Reference). This shift requires testers to analyze JSON structures, HTTP methods, and the relationship between user tokens and resource identifiers.

API-centric attacks are often subtle; they don't always trigger server crashes. Instead, they leak sensitive data slowly through unauthorized access to specific object IDs. This report aims to move beyond automated scanning and focuses on manual validation of these logical handlers.

3. Laboratory Environment Setup

A controlled penetration testing environment was established using Kali Linux. The target application was containerized to ensure isolation and reproducibility. The following steps were performed to initialize the lab:

3.1 Virtualization and Containerization

The choice of Docker allows for the rapid deployment of technical targets without affecting the host operating system's configuration. The Juice Shop image used represents a "vulnerable-by-design" architecture.

```
sudo apt update && sudo apt install docker.io -y
sudo systemctl start docker
sudo docker pull bkimminich/juice-shop
sudo docker run -d -p 3000:3000 bkimminich/juice-shop
```

3.2 Network Validation

Initial connectivity was verified using basic networking tools to ensure that port 3000 was successfully bound to the localhost interface and accepting TCP connections.

4. Tooling and Interception Proxies

The assessment utilized a multi-layered toolset to interact with the API layer effectively. While browsers are designed for consumption, security tools are designed for manipulation.

4.1 Postman and Insomnia

Postman was used as the primary Integrated Development Environment (IDE) for the API testing. It allows for the creation of request collections, handling of JWT (JSON Web Tokens) in headers, and the parameterization of variables for rapid testing of different user IDs.

```
# Typical Postman Header Setup
Content-Type: application/json
Authorization: Bearer {{global_jwt_token}}
```

4.2 cURL (Client URL)

For command-line automation and scripting, cURL was utilized. It is the gold standard for verifying vulnerabilities in a repeatable way that can be shared with development teams for patching verification.

5. API Endpoint Enumeration and Mapping

Before launching attacks, a full map of the API surface was created. By observing the traffic during a normal user session, the following RESTful patterns were identified:

- **Auth:** POST /rest/user/login
- **User Profile:** GET /api/Users/
- **Inventory:** GET /rest/products/search
- **System Info:** GET /api/system/configuration

6. Testing API1: Broken Object Level Authorization (BOLA)

BOLA is the most prevalent API security risk. It occurs when an application does not properly validate if the user requesting an object has the right to access it. In this lab, the tester identified that by simply incrementing an integer in the URL, sensitive data could be accessed.

6.1 Exploitation Steps

1. Authenticate as a low-privileged user.
2. Extract the JWT from the login response.
3. Send a GET request to /api/Users/10 (own profile).
4. Modify the request to /api/Users/1 (admin profile).

```
curl -i -H "Authorization: Bearer [TOKEN]" http://localhost:3000/
api/Users/1
```

The server incorrectly returned the administrative user's data, verifying the vulnerability. This is a failure in the permission check layer between the API gateway and the database query.

7. Testing API2: Broken Authentication

Authentication mechanisms are frequently implemented incorrectly. This includes weak password requirements, long-lived tokens, or the absence of multi-factor authentication for sensitive operations.

During the testing of the Juice Shop, the login endpoint was analyzed for its handling of incorrect attempts. It was noted that the API does not enforce strict password complexity, allowing for potential credential reuse attacks across different users on the platform.

8. Testing API4: Unrestricted Resource Consumption

Modern APIs are susceptible to Denial of Service if they do not limit how many resources a client can request. This is not just about crashing the server, but also about exhausting database connections or CPU cycles.

8.1 Rate Limiting Analysis

A stress test was conducted on the authentication endpoint. By sending 100 requests per minute from a single source, the tester looked for a "429 Too Many Requests" response code. The lack of this code indicates that an attacker could indefinitely attempt to guess credentials without being throttled.

```
# Bash script for resource exhaustion test
for i in {1..100}; do curl -X POST -s http://localhost:3000/rest/
user/login; done
```

9. Testing API8: Security Misconfiguration

Security misconfigurations often stem from the "default" state of software. This includes unpatched systems, verbose error messages, and open ports. In the API context, this often manifests as excessive data exposure in error responses.

When an invalid JSON string was sent to the Juice Shop, the server responded with a detailed stack trace including the underlying framework (Express.js) and the library versions. This information allows an attacker to search for specific CVEs relevant to those exact versions.

10. Testing API10: Unsafe Consumption of APIs

This risk occurs when an API trusts data received from another third-party API without sufficient validation. While not directly observable in the local Juiceshop without external integrations, the principle was tested by injecting "callback URLs" into fields where the server might attempt to "fetch" external data, checking for potential SSRF (Server-Side Request Forgery).

11. Remediation and Mitigation Summary

To effectively secure the API environment, a defense-in-depth approach is required. The following remediations are recommended:

- **Implement BOLA Protection:** Every request involving a resource ID must check if the current user ID (extracted from a secure, signed JWT) has the right to that resource.
- **Rate Limiting:** Use a middleware to limit requests per IP and per user account to prevent automated abuse.
- **Input Sanitization:** Implement strict JSON schema validation. If an API expects a string, it should reject an integer or an object.
- **Error Handling:** Configure the production environment to return only generic error codes (e.g., 400 Bad Request) without stack traces.

12. Conclusion

The assessment of the Juice Shop API has provided a clear picture of the risks inherent in modern web services. By mastering these ten steps—from setup to exploitation—the tester has demonstrated the ability to find and document critical authorization flaws. This capability is essential for any professional security engineer working in a VAPT or DevSecOps capacity.

13. References

1. OWASP API Security Top 10 - 2023 edition.
2. BKimminich Juice Shop Documentation.
3. Offensive Security Kali Linux Tools Documentation.