API Security Testing Manual

1. Introduction to APIs and REST

1.1 What is an API?

An **Application Programming Interface (API)** is a set of rules that allows software applications to communicate with each other. APIs define how requests and responses should be structured, enabling seamless integration between different systems.

1.2 Understanding REST APIs

Representational State Transfer (REST) is a widely used architecture for designing networked applications. RESTful APIs adhere to the following principles:

- **Statelessness**: Each request from a client to a server must contain all the necessary information.
- **Client-Server Architecture**: The client and server operate independently.
- **Cacheability**: Responses must define whether they can be cached.
- **Layered System**: APIs can have multiple layers for security and scalability.
- **Uniform Interface**: Uses standard HTTP methods such as:
    - `GET` – Retrieve resources
    - `POST` – Create resources
    - `PUT` – Update resources
    - `DELETE` – Remove resources

2. API Specification and Documentation

2.1 API Security Standards

APIs need proper documentation and security controls. Some common API security specifications include:

- **Swagger (OpenAPI Specification)**: Provides a standardized way to define and document APIs.
- **GraphQL Security**: GraphQL APIs can introduce unique security challenges such as excessive data exposure.
- **JSON Schema & XML Schema**: Used to validate request and response structures.
- **OAuth & OpenID Connect**: Used for secure authentication and authorization.

Proper API documentation helps in security auditing and vulnerability detection.

---

3. API Security Testing Process

3.1 Reconnaissance: Discovering API Endpoints

The first step in testing API security is identifying available API endpoints. This can be done using:

- **Passive Discovery**
    - Checking official API documentation (Swagger, OpenAPI specs, Postman collections).
    - Inspecting network traffic using **Burp Suite**, **ZAP**, or browser developer tools.
- **Active Discovery**
    - Crawling for endpoints using **Gospider** or **waybackurls**.
    - Checking common API paths (`/api/v1/`, `/api-docs`, `/swagger.json`).
    - Probing for hidden APIs with tools like **FFUF**, **Kiterunner**.

Example: Discovering APIs from api.example.com

```
Unset
# Discover potential API paths using FFUF
ffuf -u https://api.example.com/FUZZ -w
wordlists/common-api-endpoints.txt -mc 200

# Enumerate historical API endpoints from Wayback Machine
waybackurls api.example.com | grep '/api/' | tee
discovered_api_urls.txt

# Find hidden OpenAPI/Swagger files
curl -s https://api.example.com/swagger.json
curl -s https://api.example.com/openapi.json
curl -s https://api.example.com/api-docs
```

3.2 Detailed API Security Checklist for api.example.com

🔍 **Reconnaissance**

✅ Check `/robots.txt`, `/sitemap.xml` for hidden API paths. ✅ Perform subdomain enumeration (`api.example.com`, `internal.api.example.com`). ✅ Use **Gospider** and **Wayback Machine** to find old API versions. ✅ Probe with **FFUF** for hidden directories (`/v1/`, `/private/`, `/internal/`). ✅ Look for API documentation on `/swagger.json`, `/api-docs`, `/openapi.json`.

🛠️ Fuzzing & Endpoint Testing

✅ Use **FFUF** to brute-force API endpoints. ✅ Extract and analyze Swagger/OpenAPI definitions for more endpoints. ✅ Attempt parameter fuzzing with invalid inputs (`null`, `true/false`, long strings, SQL payloads). ✅ Test for rate-limiting using repeated requests (`Burp Suite Intruder`, `Turbo Intruder`). ✅ Enumerate HTTP methods (`GET`, `POST`, `PUT`, `DELETE`) for each endpoint.

🔐 Authentication & Authorization Testing

✅ Verify API authentication methods (`API Key`, `JWT`, `OAuth`, `Basic Auth`). ✅ Attempt access with expired or modified JWT tokens. ✅ Test **IDOR (Insecure Direct Object Reference)** by modifying resource IDs. ✅ Try accessing **admin-only** or **privileged** endpoints.

🛡️ Security Testing

✅ Test for **SQL Injection** (`sqlmap`). ✅ Check for **Cross-Site Scripting (XSS)** vulnerabilities. ✅ Perform **Server-Side Request Forgery (SSRF)** testing. ✅ Analyze CORS configurations to detect misconfigurations (`curl -I -H "Origin: evil.com"`). ✅ Check for excessive data exposure in API responses (`passwords`, `tokens`, `debug info`).

🚀 Load & Rate-Limiting Tests

✅ Perform **brute-force attacks** to check rate limits. ✅ Simulate high traffic to test API throttling (`Slowloris`, `ab`, `siege`). ✅ Inspect API logging behavior under load.

📄 Error Handling & Data Leakage

✅ Look for stack traces and verbose error messages in responses. ✅ Analyze **500 Internal Server Errors** for security misconfigurations. ✅ Detect sensitive data exposure in API responses (`email`, `tokens`, `debug data`).

---

4. Reporting and Remediation

After testing, create a report outlining:

- **API endpoint details**
- **Vulnerabilities found** (e.g., authentication flaws, rate-limiting issues)
- **Risk rating** (low, medium, high, critical)
- **Recommended fixes** (e.g., implement JWT expiration, restrict CORS, enable logging)

4.1 Automating API Security Testing

To improve API security continuously:

- Implement **CI/CD API security testing** using **Nuclei, OWASP ZAP, or Burp Suite Automation**.
- Monitor API traffic using **SIEM solutions**.
- Regularly update security policies.

---

5. Conclusion

API security testing is crucial for protecting applications from attacks. This document provides a structured approach to discovering, fuzzing, and testing APIs for vulnerabilities. Using tools like **SwaggerSpy, FFUF, Burp Suite, and Nuclei**, testers can effectively identify security flaws and recommend mitigations. Always ensure **ethical testing practices** and obtain proper authorization before performing security tests.