Lab Manual: Web Application Security Testing with OWASP ZAP

Table of Contents

---

1. Introduction

This lab manual provides a comprehensive guide to conducting web application security testing using **OWASP ZAP (Zed Attack Proxy)**. It covers theoretical backgrounds, user stories, test cases, step-by-step procedures, and reporting with recommended actions.

2. Theoretical Background

Web Application Security

Web applications are frequently targeted by attackers due to their reliance on user input and complex architectures. Security testing is essential to identify vulnerabilities and protect sensitive information.

Vulnerabilities

A vulnerability is a weakness that can be exploited by threats to gain unauthorized access or cause harm.

3. Types of Vulnerabilities

Common Vulnerabilities

- **SQL Injection**: Manipulating SQL queries through input fields.
- **Cross-Site Scripting (XSS)**: Injecting scripts into web pages.
- **Cross-Site Request Forgery (CSRF)**: Forcing users to execute unwanted actions.
- **Insecure Direct Object References**: Accessing unauthorized resources by manipulating URLs.
- **Security Misconfiguration**: Inadequate security settings.

OWASP Top 10 Vulnerabilities

1. **Injection**
2. **Broken Authentication**
3. **Sensitive Data Exposure**
4. **XML External Entities (XXE)**
5. **Broken Access Control**
6. **Security Misconfiguration**
7. **Cross-Site Scripting (XSS)**
8. **Insecure Deserialization**
9. **Using Components with Known Vulnerabilities**
10. **Insufficient Logging and Monitoring**

4. User Stories

1. **As a user**, I want to securely log in to my account to protect my personal information from unauthorized access.
2. **As an admin**, I want to ensure that unauthorized users cannot access administrative functions to maintain application integrity.
3. **As a developer**, I want to receive alerts for vulnerabilities so I can fix them before deployment to enhance security.

5. Test Cases

Comprehensive Test Cases

1. **Input Validation**
   - **Objective**: Test how the application handles malicious input.
   - **Implementation**: Send various payloads (e.g., `<script>alert('XSS')</script>`, SQL injection strings) through input fields.
2. **Authentication**
   - **Objective**: Verify that unauthorized access is denied.
   - **Implementation**: Attempt to access restricted areas without valid credentials.
3. **Session Management**
   - **Objective**: Check for session fixation and expiration issues.
   - **Implementation**: Log in, manipulate session tokens, and access user accounts.
4. **Error Handling**
   - **Objective**: Ensure errors do not expose sensitive information.

   - **Implementation**: Trigger errors and observe responses for sensitive data leakage.
5. **Access Control**
   - **Objective**: Verify proper access controls are in place.
   - **Implementation**: Attempt to access resources that should be restricted.

6. Setting Up OWASP ZAP

Install OWASP ZAP

1. Download OWASP ZAP from the official website.
2. Follow the installation instructions for your operating system.

Getting the API Key

1. **Open OWASP ZAP**: Start the application.
2. **Access API Key**:
   - Go to **Tools** > **Options**.
   - Select **API** on the left.
   - Generate or view your API key, and note it down for your Python script.

Inserting Target URL

1. **Choose Target URL**: Decide on the target URL for testing, e.g., `http://targetwebsite.com`.

7. Step-by-Step Implementation

Passive Scanning

1. **Open the Target URL**: Use the ZAP interface to navigate to your target application.
2. **Monitor Traffic**: ZAP automatically performs passive scans as you browse.
3. **Analyze Alerts**: Check for vulnerabilities identified in real-time.

Active Scanning

1. **Spider the Target**: Use ZAP's spidering feature to crawl the application.
2. **Start Active Scan**: Initiate the active scan on the discovered URLs.

Python Code Implementation

```
import time
from zapv2 import ZAPv2

# Define your API key and ZAP proxy
api_key = "ieoq1svrkl3t6uplafk1t0pe1e"
```

```
zap = ZAPv2(apikey=api_key, proxies={'http': 'http://127.0.0.1:8080',
'https': 'http://127.0.0.1:8080'})

# Target application URL
target_url = "http://targetwebsite.com"

# Passive Scan
print("Starting passive scan...")
zap.urlopen(target_url)
time.sleep(2)  # Allow time for the page to load
print("Passive scan in progress...")

# Wait for passive scan to finish
time.sleep(10)

# Check passive scan results
passive_alerts = zap.core.alerts()
print("Passive Scan Results:")
for alert in passive_alerts:
    print(f'URL: {alert["url"]}, Risk Level: {alert["risk"]}, Description:
{alert["alert"]}')

# Active Scan
print("Starting active scan...")
scan_id = zap.spider.scan(url=target_url)

# Wait for the spider scan to finish
while int(zap.spider.status(scan_id)) < 100:
    print(f'Spider progress: {zap.spider.status(scan_id)}%')
    time.sleep(2)

# Now start the active scan
active_scan_id = zap.ascan.scan(url=target_url)

# Wait for the active scan to finish
while int(zap.ascan.status(active_scan_id)) < 100:
    print(f'Active scan progress: {zap.ascan.status(active_scan_id)}%')
    time.sleep(2)

# Check active scan results
active_alerts = zap.core.alerts()
print("Active Scan Results:")
for alert in active_alerts:
    print(f'URL: {alert["url"]}, Risk Level: {alert["risk"]}, Description:
{alert["alert"]}')

print("Scanning completed.")
```

8. Analyzing Alerts and Risk Levels

Alerts

Alerts are generated based on the findings from passive and active scans. Each alert contains:

- **URL**: The affected resource.
- **Risk Level**: Classified as Low, Medium, High, or Informational.
- **Description**: Details about the nature of the vulnerability.

Risk Levels

- **Low**: Minor issues, unlikely to be exploited.
- **Medium**: Moderate impact, can be exploited under certain conditions.
- **High**: Significant impact, likely to be exploited.

9. Reporting Steps and Corresponding Actions

Generating Reports

1. **Collect Alerts**: After scanning, gather all alerts.
2. **Categorize Alerts**: Sort by risk level and type.
3. **Document Findings**: Create a report that includes:
    - Overview of scans performed.
    - List of vulnerabilities found.
    - Risk levels and remediation suggestions.

Example Report Structure

- **Executive Summary**: A brief overview of findings.
- **Vulnerability Details**: List vulnerabilities with descriptions and risk levels.
- **Recommendations**: Actions to remediate each vulnerability.
- **Conclusion**: Final thoughts on the security posture.

Corresponding Actions

- **For High-Risk Vulnerabilities**: Immediate remediation is required. Consider patching the application or implementing additional security controls.
- **For Medium-Risk Vulnerabilities**: Schedule a fix as part of the next deployment cycle.
- **For Low-Risk Vulnerabilities**: Review during regular maintenance cycles; assess whether they pose a legitimate risk.

10. Conclusion

This lab manual provides a detailed guide for using OWASP ZAP to conduct web application security testing. By following these steps, you can effectively identify and address vulnerabilities, enhancing the security posture of your web applications. Always ensure that testing is conducted ethically and with appropriate permissions.