Web Application Pentesting

Index

Introduction

Web application penetration testing is a methodical process of identifying, analyzing, and exploiting vulnerabilities within a web application to secure it from potential threats. This document covers key aspects of web application pentesting, including the OWASP Top 10 Web Risks, a comprehensive pentesting checklist, and detailed insights into authentication, authorization, session management, file security, web application firewalls, and essential tools.

OWASP Top 10 Web Risks

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications:

1. **Injection**
    - **Example**: SQL Injection
    - **Scenario**: Exploiting user input fields to execute arbitrary SQL commands.
    - **Mitigation**: Use prepared statements and parameterized queries.
2. **Broken Authentication**
    - **Example**: Weak password policy
    - **Scenario**: Attacker gains access using brute force or credential stuffing.
    - **Mitigation**: Implement strong password policies, multi-factor authentication (MFA).
3. **Sensitive Data Exposure**
    - **Example**: Unencrypted data transmission
    - **Scenario**: Attacker intercepts data in transit.
    - **Mitigation**: Use TLS/SSL for data transmission.
4. **XML External Entities (XXE)**
    - **Example**: Parsing XML input
    - **Scenario**: Attackers exploit vulnerable XML parsers.
    - **Mitigation**: Disable XML external entity processing.
5. **Broken Access Control**
    - **Example**: Missing function-level access control
    - **Scenario**: Unauthorized access to restricted functionalities.
    - **Mitigation**: Enforce proper access control checks.

6. **Security Misconfiguration**
    - **Example**: Default configurations
    - **Scenario**: Using default credentials or settings.
    - **Mitigation**: Regularly review and update configurations.
7. **Cross-Site Scripting (XSS)**
    - **Example**: Reflected XSS
    - **Scenario**: Injecting malicious scripts into web pages.
    - **Mitigation**: Sanitize and validate user inputs.
8. **Insecure Deserialization**
    - **Example**: Deserialization of untrusted data
    - **Scenario**: Remote code execution.
    - **Mitigation**: Avoid deserializing untrusted data.
9. **Using Components with Known Vulnerabilities**
    - **Example**: Outdated libraries
    - **Scenario**: Exploiting known vulnerabilities in third-party components.
    - **Mitigation**: Regularly update and patch software components.
10. **Insufficient Logging & Monitoring**
    - **Example**: Lack of logging
    - **Scenario**: Undetected security breaches.
    - **Mitigation**: Implement comprehensive logging and monitoring.

Web Application Pentesting Checklist

1. **Pre-engagement Interactions**
    - Define the scope and objectives.
    - Gather necessary permissions.
2. **Information Gathering**
    - Identify the web application and server technologies.
    - Enumerate subdomains, directories, and files.
    - Identify user inputs and data entry points.
3. **Vulnerability Identification**
    - Test for OWASP Top 10 vulnerabilities.
    - Analyze authentication and authorization mechanisms.
    - Test session management.
4. **Exploitation**
    - Attempt to exploit identified vulnerabilities.
    - Document successful exploitation steps.
5. **Post-Exploitation**
    - Assess the impact of exploited vulnerabilities.
    - Gather additional information for further exploitation.
6. **Reporting**
    - Document findings in a detailed report.
    - Provide remediation recommendations.

Authentication & Authorization

- **Authentication**: Verify the identity of users.
    - **Example**: Brute force attack on login page.
    - **Mitigation**: Implement rate limiting, account lockout mechanisms.
- **Authorization**: Control access to resources.
    - **Example**: Vertical privilege escalation.
    - **Mitigation**: Implement role-based access control (RBAC).

Session Management

- **Objective**: Securely manage user sessions.
- **Example**: Session fixation attack.
- **Mitigation**: Use secure session cookies, regenerate session IDs upon login.

File Security

- **Objective**: Protect files from unauthorized access.
- **Example**: Unrestricted file upload leading to remote code execution.
- **Mitigation**: Implement file validation, restrict file types, and store files outside the web root.

Web Application Firewalls (WAF)

- **Objective**: Filter and monitor HTTP traffic.
- **Example**: Blocking common attack patterns.
- **Tools**: ModSecurity, AWS WAF.
- **Scenario**: Deploying a WAF to block SQL injection attempts.

Tools

- **BurpSuite**
    - **Usage**: Comprehensive web vulnerability scanner.
    - **Example**: Intercepting and modifying HTTP requests.
- **Sqlmap**
    - **Usage**: Automated SQL injection tool.
    - **Example**: Exploiting SQL injection to extract database information.
- **wafw00f**

- **Usage**: Identify web application firewalls.
- **Example**: Detecting WAFs to plan bypass techniques.

Lab Guide Example

1. **Setting up a Lab Environment**
   - Install a vulnerable web application (e.g., DVWA, OWASP Juice Shop).
   - Configure the environment with tools (BurpSuite, Sqlmap, wafw00f).
2. **Running Tests**
   - Perform information gathering using BurpSuite.
   - Identify and exploit SQL injection with Sqlmap.
   - Detect the presence of a WAF using wafw00f.
3. **Document Findings**
   - Record vulnerabilities and exploitation steps.
   - Provide detailed remediation strategies.

Step-by-Step Lab Guide for Web Application Penetration Testing

Prerequisites

- Basic understanding of web application security concepts.
- Installed tools: BurpSuite, Sqlmap, wafw00f.
- A test environment with a vulnerable web application (e.g., DVWA or OWASP Juice Shop).

Step 1: Setting Up the Lab Environment

1. **Install a Vulnerable Web Application**
    - **Download DVWA (Damn Vulnerable Web Application)**

git clone https://github.com/digininja/DVWA.git

cd DVWA

    - **Configure DVWA**
        - Follow the DVWA setup guide to configure the web application.
    - **Start the Web Server**

sudo service apache2 start

sudo service mysql start

2. **Install and Configure BurpSuite**
    - **Download BurpSuite** from the official website.
    - **Install BurpSuite** following the installation instructions for your OS.
    - **Configure Browser to use BurpSuite as a Proxy**
        - Set the browser proxy to 127.0.0.1:8080.
3. **Install Sqlmap**
    - **Download Sqlmap**

git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev

cd sqlmap-dev

4. **Install wafw00f**
    - **Download wafw00f**

pip install wafw00f

Step 2: Information Gathering

1. **Using BurpSuite for Information Gathering**
    - **Open BurpSuite** and navigate to the **Proxy** tab.
    - **Browse the Target Application** through the configured browser to capture requests.
    - **Analyze Captured Requests** in BurpSuite to identify parameters, endpoints, and technologies.

Step 3: Vulnerability Identification

1. **Testing for SQL Injection with Sqlmap**
    - Identify a URL with a parameter (e.g., http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit).
    - **Run Sqlmap**

python sqlmap.py -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" --batch --dbs

    - **Analyze Sqlmap Output** for database information and potential vulnerabilities.
2. **Using BurpSuite for XSS Testing**
    - **Intercept Requests** in BurpSuite containing input fields.
    - **Inject XSS Payloads** (e.g., <script>alert(1)</script>) into input fields.
    - **Observe Responses** to see if the payload is executed.
3. **Detecting Web Application Firewalls with wafw00f**
    - **Run wafw00f**

wafw00f http://localhost

    - **Analyze Output** to determine if a WAF is present and its type.

Step 4: Exploitation

1. **Exploiting SQL Injection**
    - **Extract Database Information** using Sqlmap.

python sqlmap.py -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" --batch --current-db

2. **Exploiting XSS**
    - **Craft Persistent XSS Payloads** that store scripts in the application.
    - **Inject Payloads** into fields that store data persistently (e.g., comment sections).
    - **Verify Execution** by navigating to the affected pages.

Step 5: Post-Exploitation

1. **Assessing Impact**
    - **Document Database Information** retrieved through SQL injection.
    - **Identify Stored Scripts** and their execution contexts for XSS.
2. **Gathering Additional Information**
    - **Explore Further Injection Points** and input fields.
    - **Document Additional Findings** and potential security risks.

Step 6: Reporting

1. **Document Findings**
    - **Create a Detailed Report** including all identified vulnerabilities, exploitation steps, and evidence (screenshots, logs).
    - **Structure Report** with sections for each vulnerability type (SQLi, XSS, etc.).
2. **Provide Remediation Recommendations**
    - **SQL Injection**
        - Use prepared statements and parameterized queries.
        - Sanitize and validate all user inputs.
    - **XSS**
        - Sanitize and encode user inputs.
        - Implement Content Security Policy (CSP).
    - **General Recommendations**
        - Regularly update and patch software components.
        - Implement robust authentication and authorization mechanisms.
        - Enable comprehensive logging and monitoring.

Conclusion

This step-by-step lab guide provides a structured approach to web application penetration testing, focusing on key vulnerabilities and their exploitation using essential tools. Regular practice in a controlled environment like this enhances the skills necessary to secure real-world applications against various cyber threats.