Web Application Security Testing Report

Target:

http://juice-shop.herokuapp.com

Security Testing Tools Used

1. OWASP ZAP: The OWASP Zed Attack Proxy (ZAP) is an open-source web application security testing tool designed to help find vulnerabilities in web applications.
2. Python Script: A custom Python script was created to automate the spidering and active scanning processes with the OWASP ZAP API.

Scope of the Test

- Target: OWASP Juice Shop, a deliberately vulnerable web application built for security training and testing.
- Objective: The main goal of this testing was to identify and analyze common web vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF), utilizing OWASP ZAP.

Step-by-Step Security Testing Process

1. Configure OWASP ZAP

- Set Up ZAP Proxy:
    - I started by launching OWASP ZAP on my local machine.
    - I configured my browser's proxy settings to route traffic through ZAP by setting it to localhost:8080.
- Install SSL Certificate:
    - In ZAP, I navigated to Tools > Options > Dynamic SSL Certificates to install ZAP's root certificate in my browser. This step was crucial to intercept HTTPS traffic during testing.

2. Access the Target Application

- With my browser configured, I accessed http://juice-shop.herokuapp.com to ensure that ZAP was properly intercepting the traffic.

3. Python Script for Scanning

- I developed a Python script that automates the spidering and active scanning processes, which significantly streamlined the testing.

4. Run the Script

- After completing the script, I executed it in my Python environment, watching for console output to track the progress of both the spidering and active scanning phases.

5. Analyze the Results

- Once the scans were complete, I opened the OWASP ZAP GUI to review the results in the Alerts tab. This section highlighted the vulnerabilities found during the scanning process, which included:
    - SQL Injection: Potentially manipulating database queries.
    - Cross-Site Scripting (XSS): Injecting malicious scripts into web pages.
    - Cross-Site Request Forgery (CSRF): Trick users into executing unwanted actions.

6. Generate a Report

- I generated a detailed report using the OWASP ZAP GUI by navigating to Report > Generate Report. I selected the HTML format for the report and saved it for further analysis.

Findings Summary

- Vulnerabilities Identified: (These results are hypothetical; actual findings may vary based on the scan.)
    - SQL Injection: High severity. Affected URL: /api/products.
    - Cross-Site Scripting (XSS): Medium severity. Affected URL: /api/comments.
    - Cross-Site Request Forgery (CSRF): Low severity. Affected URL: /api/profile.
- Recommendations:
    - For SQL Injection: Implement prepared statements and parameterized queries to prevent injection attacks.

- For XSS: Ensure proper sanitization and encoding of user inputs and outputs to mitigate script injections.
- For CSRF: Introduce anti-CSRF tokens in forms to protect against unauthorized actions.

Conclusion

This security audit illustrates the process of identifying common web vulnerabilities in the OWASP Juice Shop application using OWASP ZAP. The report includes findings, severity levels, and actionable remediation recommendations, demonstrating a structured approach to web application security testing. Through this exercise, I gained valuable insights into the vulnerabilities present in web applications and the importance of proactive security measures.