Web Application Security

Project Overview: This project focuses on performing a vulnerability assessment on WebGoat which is an intentionally insecure web application maintained by OWASP to teach security professionals about common web vulnerabilities.

Objective: Learn about common web application vulnerabilities by analyzing a simple web application. This task will help you understand how attackers can exploit weaknesses in web applications.

TOOLS USED AND SETUP:

1. WebGoat: A deliberately vulnerable web application used as a hands-on platform for learning and practicing common web security vulnerabilities
2. OWASP ZAP: An open-source security tool used to identify, scan and exploit web application vulnerabilities through automated and manual testing methods
3. Kali Linux: Operating system used to host testing environment and security tools during assessment process.

The WebGoat application was hosted locally and OWASP ZAP was configured as an intercepting proxy to scan traffic and identify vulnerabilities.

METHODOLOGY

1. Install and configure WebGoat on a virtual machine via Docker.
2. Launch OWASP ZAP and configure it to proxy traffic through the browser.
3. Browse the WebGoat application and intercept requests.
4. Identify vulnerabilities based on input/output behavior.
5. Exploit the vulnerabilities manually.
6. Record findings and provide mitigation steps for each issue.

FINDINGS

1. Vulnerability 1 – SQL Injection

• Description: Login form is vulnerable to SQL Injection.

- Discovery method: Injected ‘ OR ‘1’ = ‘1 in the username field.
- Exploitation process: Intercepted login request using ZAP, modified input to ‘ OR ‘1’ = ‘1 for the username and was able to login and bypass authentication
- Risk: Gives unauthorized access to data and credentials, Database altering.
- Mitigation:
  a) Validate and sanitize all user inputs: Ensure that all data received from users is thoroughly validated and sanitized to remove potentially dangerous characters.
  b) Implement parameterized queries (Prepared statements): Use secure coding techniques such as parameterized queries or prepared statements to separate SQL logic from user inputs preventing the injection of malicious code.
  c) Avoid dynamic SQL Construction with user input: Refrain from directly inputting user-provided data into SQL queries.
  d) Implement least privilege for database access: Configure the database to grant the application only the necessary permissions such as read-only access where applicable which helps reduce impact of successful injections.

2. Vulnerability 2 – Cross-Site Request Forgery
   - Description: Email change form. The application does not implement anti-CSRF tokens for form submissions which allows attackers to perform unauthorized actions on behalf of authenticated users.
   - Discovery: Detected via manual review in OWASP ZAP.
   - Exploitation Process:
     i) Created a malicious form targeting a vulnerable endpoint ii) Hosted it on another domain iii) When the page is visited by an individual, the form auto-submitted and changed details.
   - Mitigation:
     a) Implement Anti-csrf Tokens
     b) Use SameSite Cookies
     c) Avoid GET requests for state-changing actions
     d) Confirm user intent with re-authentication or CAPTCHA

3. Vulnerability 3 – Cross-Site Scripting (XSS)
   - Description: Reflected XSS in search fields
   - Discovery: Entered <script>alert('XSS')</script> • Risk: Session hijacking, phishing, malware injection.
   - Exploitation process:
     i. Input malicious script into search field.
     ii. Script was executed in browser.
   - Mitigation:
     i. Sanitize user inputs ii. Encode output data before rendering iii. Implement a strong Content Security Policy (CSP).