SECURITY ASSESSMENT

OWASP Juice Shop Penetration Test Report

Submitted to:
DEPI - Ministry of Communications and Information Technology
Pentester Name:

- Mohammad Abd El-Fattah
- Saif adDin Samir Shenawi
- Amr Khaled Zakaria
- David Milad
- Eslam Khaled Ali

Date of Testing: October 4-22, 2024
Date of Report Delivery: October 22, 2024

Table of Contents

Security Engagement Summary

Engagement Overview

The engagement involved assessing the security of the Juice Shop application. The goal was to identify and document vulnerabilities in various aspects of the web application. The engagement was requested to improve the security posture and protect against potential exploitation. It was carried out by a security analyst as part of a periodic vulnerability assessment.

Scope

The assessment focused on various components of the Juice Shop application, including authentication, file upload mechanisms, cryptographic functions, and exposure to common web vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection, and insecure deserialization.

Executive Risk Analysis

Risk Level: High

The overall risk rating is high due to multiple critical and high-risk vulnerabilities identified, such as exposure of sensitive information, weak password policies, and outdated cryptographic practices.

Executive Recommendation

Remediation efforts are warranted immediately, prioritizing the highest-risk vulnerabilities such as exposure of sensitive data and insecure file upload mechanisms. Implement proper access controls, enforce strong password policies, and replace deprecated cryptographic algorithms.

Significant Vulnerability Summary

High Risk Vulnerabilities

1. **Broken Access Control in the Admin Section**
   Unauthorized access to the admin section was achieved by exploiting SQL injection vulnerabilities in the login page, leading to full administrative access.
2. **Exposure of Sensitive Information via FTP Directory**
   An unprotected FTP directory exposed a KeePass database containing sensitive credentials, making the application vulnerable to credential theft and unauthorized access.
3. **Deprecated Interface and Improper File Type Validation in File Upload**
   The file upload functionality allowed .xml files, creating a risk of XML External Entity (XXE) attacks despite interface restrictions suggesting only .pdf and .zip files were allowed.

Medium Risk Vulnerabilities

1. **DOM-Based Cross-Site Scripting (XSS)**
   The search functionality was vulnerable to DOM-based XSS attacks, allowing attackers to inject malicious scripts by manipulating URL parameters.
2. **Improper Error Handling on Image URL Input in Profile Page**
   The application exposed internal file paths when invalid inputs were provided in the image URL field, revealing sensitive server information.
3. **Lack of Rate-Limiting for Login Attempts**
   The absence of rate-limiting on login attempts made the application susceptible to brute-force attacks, increasing the risk of unauthorized access.

Low Risk Vulnerabilities

1. **Security Policy Exposure (Miscellaneous)**
   Security policy files were accessible through common paths, revealing information about security practices and challenge-solving guidelines.
2. **Insecure Configuration of Error Messages**
   Error messages provided detailed internal server information that could aid attackers in understanding the application’s structure and potential weak points.

Significant Vulnerability Detail

Cross-Site Scripting (XSS) Vulnerability

RISK LEVEL: MEDIUM

Vulnerability Detail:
A Cross-Site Scripting (XSS) vulnerability was identified in the Juice Shop application, allowing the injection of an iframe that could embed unauthorized third-party content, such as a SoundCloud player.

Risk Level Assessment: Medium

Discussion (Executive Summary):
The vulnerability was discovered during a security assessment focused on the “Bonus Payload” challenge, which aimed to test the application’s defenses against XSS attacks. Manual testing techniques were employed, including crafting and executing an XSS payload to insert an iframe. The successful injection demonstrated the application’s susceptibility to such attacks.

Evidence of Validation:
The URL used for the injection was:
`http://127.0.0.1:3000/#/search?q=%3Ciframe%20width%3D%22100%25%22%20height%3D%22166%22%20scrolling%3D%22no%22%20frameborder%3D%22no%22%20allow%3D%22autoplay%22%20src%3D%22https:%2F%2Fw.soundcloud.com%2Fplayer%2F%3Furl%3Dhttps%253A%2F%2Fapi.soundcloud.com%2Ftracks%2F771984076%26color%3D%2523ff5500%26auto_play%3Dtrue%26hide_related%3Dfalse%26show_comments%3Dtrue%26show_user%3Dtrue%26show_reposts%3Dfalse%26show_teaser%3Dtrue%22%3E%3C%2Fiframe%3E`.

Probability of Exploit/Attack:
The likelihood of exploitation is moderate, as the vulnerability can be exploited by any user who can input data into the application, potentially leading to significant security risks.

Impact:
If exploited, this vulnerability could affect all users of the Juice Shop application, leading to compromised user trust, potential data breaches, and financial losses due to phishing or malware distribution.

Potential Remediation:

- **Content Security Policy (CSP):** Implement a strict CSP to limit which external sources can be embedded.
- **Input Sanitization:** Ensure thorough sanitization of user inputs to prevent HTML elements from being injected into the DOM.

Weak Password Policy Vulnerability

RISK LEVEL: HIGH

Vulnerability Detail:
The Juice Shop application allows the use of a weak password (`admin123`) for the administrator account, making it highly susceptible to brute-force attacks.

Risk Level Assessment: High

Discussion (Executive Summary):
This vulnerability was identified during a security assessment focused on the “Password Strength” challenge. The assessment revealed that the administrator’s account could be easily compromised due to the weak password policy. A brute-force attack was successfully executed using Burp Suite, confirming the vulnerability.

Evidence of Validation:
A brute-force attack was conducted using Burp Suite’s Intruder module, successfully gaining access with the password `admin123`. Screenshots of the attack setup and successful login are available.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as the weak password can be easily guessed or brute-forced by attackers.

Impact:
If exploited, this vulnerability could grant unauthorized access to the administrator account, allowing attackers to modify sensitive data, disable security controls, or launch further attacks on users.

Potential Remediation:

- **Strong Password Policies:** Enforce complexity requirements, including uppercase and lowercase letters, numbers, and special characters.
- **Rate-Limiting:** Implement rate-limiting for login attempts to mitigate brute-force attacks.
- **Multi-Factor Authentication (MFA):** Introduce MFA for administrative accounts to enhance security.

Lack of Rate-Limiting for Login Attempts

RISK LEVEL: MEDIUM

Vulnerability Detail:
The Juice Shop application does not enforce rate-limiting on login attempts, allowing unlimited guesses, which increases the risk of brute-force attacks.

Risk Level Assessment: Medium

Discussion (Executive Summary):
The absence of rate-limiting was identified during the assessment, exacerbating the risk posed by weak passwords. Attackers can continuously attempt to guess passwords without restriction.

Evidence of Validation:
The assessment demonstrated that the login interface allowed unlimited attempts, confirming the lack of protective measures.

Probability of Exploit/Attack:
The probability of exploitation is moderate, as attackers can repeatedly attempt to guess passwords without facing any barriers.

Impact:
Without rate-limiting, the application is vulnerable to continuous brute-force attacks, significantly increasing the risk of unauthorized access.

Potential Remediation:

- **Rate-Limiting:** Introduce limits on login attempts and temporarily lock accounts after a specified number of failed attempts.
- **CAPTCHA:** Consider integrating CAPTCHA to further deter automated attacks.

Exposure of Sensitive Information

RISK LEVEL: HIGH

Vulnerability Detail:
An exposed FTP directory containing a KeePass database file could lead to sensitive data being compromised.

Discussion (Executive Summary):
This vulnerability was identified through a security assessment that revealed an unsecured FTP directory. The presence of sensitive files, such as a KeePass database, poses a significant risk.

Evidence of Validation:
Access to the FTP directory confirmed the presence of the KeePass database file.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can easily access exposed directories.

Impact:
If exploited, sensitive user data could be compromised, leading to potential data breaches and loss of trust.

Potential Remediation:
Restrict access to the FTP directory and implement proper authentication mechanisms to secure sensitive files.

Deprecated Interface for File Upload

RISK LEVEL: HIGH

Vulnerability Detail:
The ability to upload .xml files poses risks of XML External Entity (XXE) attacks.

Discussion (Executive Summary):
This vulnerability was identified during testing of the file upload functionality, where .xml files could be uploaded without proper validation.

Evidence of Validation:
Testing confirmed that uploaded .xml files could be manipulated to exploit XXE vulnerabilities.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can craft malicious XML files.

Impact:
Exploitation could lead to unauthorized access to internal resources and sensitive data exposure.

Potential Remediation:
Implement strict validation on file uploads and disable the processing of external entities in XML parsers.

Broken Access Control to Admin Section

RISK LEVEL: HIGH

Vulnerability Detail:
Attempts to access admin-related subdomains returned a 403 error. However, using an SQL injection payload (`admin' OR 1==1 --`) bypassed authentication, granting admin access.

Discussion (Executive Summary):
This vulnerability was identified by testing access controls on admin subdomains and exploiting SQL injection.

Evidence of Validation:
Successful login using the SQL injection payload confirmed broken access control.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as SQL injection can be easily executed.

Impact:
Exploitation could lead to unauthorized access to sensitive admin functionalities and data.

Potential Remediation:
Implement robust access control measures and validate user input to prevent SQL injection.

Security Policy (Miscellaneous)

RISK LEVEL: LOW

Vulnerability Detail:
The security policy file was found at `/.well-known/security.txt`, highlighting the need for standardized security practices.

Discussion (Executive Summary):
This vulnerability was identified by searching for common paths where security policies are typically located.

Evidence of Validation:
Accessing the file confirmed its presence.

Probability of Exploit/Attack:
The likelihood of exploitation is low, as the information is not sensitive.

Impact:
While not critical, exposure of the security policy could provide insights to attackers.

Potential Remediation:
Consider restricting access to the security policy file or ensuring it contains minimal information.

Register as Admin

RISK LEVEL: HIGH

Vulnerability Detail:
By modifying the registration request to assign the role as “admin,” an attacker could successfully register as an admin.

Discussion (Executive Summary):
This vulnerability was identified during testing of the registration process, where role assignment could be manipulated.

Evidence of Validation:
Successful registration as an admin confirmed the security misconfiguration.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as the registration process lacks proper validation.

Impact:
Exploitation could lead to unauthorized admin access, compromising the application’s security.

Potential Remediation:
Implement strict validation on user roles during registration and ensure unique email addresses.

Forged Feedback

RISK LEVEL: MEDIUM

Vulnerability Detail:
Intercepting and modifying the feedback request allowed submission as another user, exposing an authorization flaw.

Discussion (Executive Summary):
This vulnerability was identified by testing the feedback functionality and manipulating requests.

Evidence of Validation:
Successful submission of feedback as another user confirmed the flaw.

Probability of Exploit/Attack:
The likelihood of exploitation is moderate, as it requires request interception.

Impact:
If exploited, this could lead to unauthorized actions being taken on behalf of other users.

Potential Remediation:
Implement proper authorization checks for feedback submissions.

Forged Review

RISK LEVEL: MEDIUM

Vulnerability Detail:
Editing the “author” and “message” fields in a review request enabled forging a review as another user.

Discussion (Executive Summary):
This vulnerability was identified by testing the review functionality and manipulating requests.

Evidence of Validation:
Successful submission of a forged review confirmed inadequate input validation.

Probability of Exploit/Attack:
The likelihood of exploitation is moderate, as it requires request interception.

Impact:
If exploited, this could lead to misinformation and damage to the application’s credibility.

Potential Remediation:
Implement proper input validation and user authorization checks for reviews.

DOM-Based XSS

RISK LEVEL: MEDIUM

Vulnerability Detail:
The search input was vulnerable to XSS, allowing execution of arbitrary JavaScript through injected payloads.

Discussion (Executive Summary):
This vulnerability was identified during testing of the search functionality.

Evidence of Validation:
Injecting a payload demonstrated the execution of arbitrary JavaScript.

Probability of Exploit/Attack:
The likelihood of exploitation is moderate, as it requires user interaction.

Impact:
If exploited, this could lead to session hijacking or data theft.

Potential Remediation:
Implement proper input sanitization and output encoding to prevent XSS.

Improper Error Handling on Image URL Input in Profile Page

RISK LEVEL: MEDIUM

Vulnerability Detail:
The image URL field on the profile page is vulnerable to improper error handling, which can disclose internal file paths and application structure information. This increases the risk of further attacks, such as directory traversal and local file inclusion (LFI).

Discussion (Executive Summary):
This vulnerability was identified during testing of the profile page located at `http://localhost:3000/profile`. When an invalid input (e.g., “ffff”) is provided in the image URL field, the application returns a detailed error message that exposes sensitive information about the server’s directory structure.

Evidence of Validation:
Upon submitting invalid data, the application returned an error message that included internal file paths, confirming the vulnerability. For example, the error message indicated a 500 Error with details about the request handling.

Probability of Exploit/Attack:
The likelihood of exploitation is medium, as anyone with access to the profile page can trigger the error by entering invalid data.

Impact:
If exploited, this vulnerability could provide attackers with insights into the server’s directory structure, aiding in further attacks such as directory traversal, local file inclusion, or code execution.

Potential Remediation:

- **Short-Term Fix:** Disable the display of detailed error messages to users. Configure the application to show generic error messages while logging detailed information internally. Validate all input data in the image URL field to ensure it follows the expected format.
- **Long-Term Fix:** Refactor the image upload handler to implement proper error handling and input validation. Only valid URLs or file paths should be accepted. Conduct a thorough review of error handling across the entire application to ensure no sensitive information is leaked.

Deprecated Interface and Improper File Type Validation in File Upload

RISK LEVEL: HIGH

Vulnerability Detail:
The file upload functionality on the Complaint page allows the upload of deprecated and less secure file types (e.g., .xml), bypassing restrictions that should only permit .pdf and .zip files. This could lead to security issues such as XML External Entity (XXE) attacks, local file inclusions (LFI), or other vulnerabilities.

Discussion (Executive Summary):
This vulnerability was identified during an assessment of the file upload functionality at `http://localhost:3000/#/complain`. Although the interface indicates that only .pdf and .zip files are allowed, an examination of the source code revealed that .xml files are also accepted. This discrepancy between the user interface and backend logic allows attackers to bypass file type restrictions and upload insecure formats.

Evidence of Validation:
Inspecting the source code showed that the allowed MIME types include:

```
allowedMimeType: [
  'application/pdf',
  'application/xml',
  'text/xml',
  'application/zip',
  'application/x-zip-compressed',
  'multipart/x-zip'
]
```

This confirms that XML files can be uploaded despite UI restrictions.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can easily upload malicious .xml files by exploiting the file upload functionality.

Impact:
If exploited, this vulnerability could lead to:

- **XXE (XML External Entity) Attacks:** Malicious XML files could exploit vulnerable XML parsers to access sensitive server files or exfiltrate data.
- **File Inclusion Attacks:** Improper handling of uploaded files could lead to local or remote file inclusion vulnerabilities.
- **Inconsistent Security Policy:** Discrepancies between the frontend and backend create confusion and weaken the overall security of the application.

Potential Remediation:

- **Short-Term Fix:** Implement proper file type validation to restrict uploads to only .pdf and .zip files. Disable the acceptance of deprecated file types like .xml.
- **Long-Term Fix:** Refactor the file upload handler to ensure that only secure file types are accepted. Conduct a thorough review of the file upload functionality to align the frontend and backend security policies.

Cross-Site Scripting (XSS) via Image Parameter Manipulation

RISK LEVEL: HIGH

Vulnerability Detail:
The Deluxe Membership page allows attackers to manipulate image sources via the `testDecal` query parameter, potentially redirecting users to malicious sites or injecting harmful content into the application.

Discussion (Executive Summary):
This vulnerability was identified on the Deluxe Membership page at `http://localhost:3000/#/deluxe-membership`. The `testDecal` parameter in the URL allows arbitrary input to change the logo image. An attacker can manipulate this parameter to load external resources, leading to malicious redirects or further exploitation due to a lack of input validation and improper sanitization.

Evidence of Validation:
By modifying the URL to include a malicious payload, such as:

http://localhost:3000/#/deluxe-membership?testDecal=../../../../../redirect?to=https://dummyimage.com/600x400/000/fff?x=https://github.com/juice-shop/juice-shop

the logo image can be replaced with an external image or redirect users to a malicious destination.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can easily craft a malicious URL with a manipulated `testDecal` parameter, taking advantage of the lack of input validation.

Impact:
If exploited, this vulnerability could allow attackers to replace the logo with an external image or redirect users to malicious sites, facilitating phishing attacks. Additionally, attackers could load external scripts, leading to stored XSS attacks and potentially compromising user sessions.

Potential Remediation:

- **Input Validation:** Implement strict validation and sanitization of the `testDecal` parameter to prevent arbitrary input.
- **Content Security Policy (CSP):** Enforce a CSP to restrict the sources from which images and scripts can be loaded.
- **User Education:** Inform users about the risks of clicking on unexpected links or images.

Exposure of Sensitive Credentials via Accessible FTP Directory

RISK LEVEL: HIGH

Vulnerability Detail:
The Juice Shop application has an unprotected FTP directory accessible at `http://localhost:3000/ftp`, which contains a KeePass database file (`incident-support.kdbx`). This vulnerability allows attackers to access sensitive credentials, including the email and password for an administrative account (support@juice-sh.op), using password-cracking techniques.

Discussion (Executive Summary):
A critical vulnerability was discovered during a directory brute-force scan using tools like `dirb`, which revealed an exposed FTP folder containing the KeePass database file. By utilizing tools such as `keepass2john` and `john`, the database was cracked, exposing sensitive credentials for a support team Gmail account. This vulnerability poses a significant risk, as it allows unauthorized access to sensitive information and could potentially compromise the application's administrative account.

Evidence of Validation:
The exposed FTP directory was confirmed by navigating to `http://localhost:3000/ftp` and downloading the `incident-support.kdbx` file. The password for the database was successfully cracked using:

```
keepass2john incident-support.kdbx > support.txt
john support.txt
```

The cracked password was revealed as `support2022!`, allowing access to the sensitive credentials.

Probability of Exploit/Attack:
The likelihood of exploitation is critical, as attackers can easily perform a directory brute-force scan to discover the FTP directory and download the KeePass file.

Impact:
If exploited, this vulnerability could lead to unauthorized access to sensitive credentials, including the support email account (support@juice-sh.op). This access could facilitate further exploitation, data leakage, and potential compromise of the entire application.

Potential Remediation:

- **Restrict FTP Access:** Secure the FTP directory with proper authentication mechanisms to prevent unauthorized access.
- **Remove Sensitive Files:** Ensure that sensitive files, such as KeePass databases, are not stored in publicly accessible directories.

Insecure Deserialization and Denial of Service (DoS) via API Manipulation

RISK LEVEL: HIGH

Vulnerability Detail:
The Juice Shop application is vulnerable to insecure deserialization and Denial of Service (DoS) attacks through the `/orders` API endpoint. This endpoint accepts arbitrary JSON data, including the `orderLinesData` attribute, which can be exploited to execute malicious payloads.

Discussion (Executive Summary):
This vulnerability was identified by discovering the Swagger API documentation at `http://localhost:3000/api-docs` using GoBuster. The `/orders` endpoint allows adding new orders and accepts JSON data, including the `orderLinesData` parameter. By manipulating this parameter, attackers can inject payloads designed to cause infinite loops or excessive backtracking, leading to server unresponsiveness.

Evidence of Validation:
Testing the API endpoint revealed that the `orderLinesData` parameter allows arbitrary JSON injection. For example, a payload designed to create an infinite loop was crafted as follows:

```
{ "orderLinesData": "(function dos() { while(true); })()"}
```

After executing this payload, the server became unresponsive for about 2 seconds before recovering and displaying an internal server error message.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can easily craft malicious payloads and execute them through the Swagger UI without any input validation.

Impact:
If exploited, this vulnerability could lead to Denial of Service conditions, making the application unavailable to legitimate users. Additionally, it could allow for further exploitation through insecure deserialization, potentially compromising the application’s integrity.

Potential Remediation:

- **Input Validation:** Implement strict validation and sanitization of the `orderLinesData` parameter to prevent arbitrary JSON injection.
- **Rate Limiting:** Introduce rate limiting on API requests to mitigate the impact of DoS attacks.
- **Error Handling:** Improve error handling to prevent detailed error messages from being exposed to users.

Unvalidated Redirects via Outdated Allowlist

RISK LEVEL: HIGH

Vulnerability Detail:
This vulnerability involves exploiting unvalidated redirects and the use of an outdated allowlist in the Juice Shop web application. Attackers can manipulate the URL to redirect users to unapproved or unintended cryptocurrency addresses.

Discussion (Executive Summary):
The vulnerability was identified by analyzing the QR code functionality for cryptocurrency transactions. The JavaScript code in `main.js` revealed hard-coded cryptocurrency addresses and a redirect function based on user selection. The `showBitcoinQrCode` function demonstrates this:

```
showBitcoinQrCode() {  this.dialog.open({

    data: {
      data: "bitcoin:1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
      url: "./redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
      address: "1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
      title: "TITLE_BITCOIN_ADDRESS"}})}
```

Evidence of Validation:
By modifying the redirect URL parameter, the following URL was accessed:

http://192.168.1.108:3000/redirect?to=https://www.blockchain.com/explorer/addresses/btc/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm

This manipulation confirmed that the application allowed redirects to unintended addresses.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can easily manipulate the redirect URL without any validation checks.

Impact:
If exploited, this vulnerability could lead to users being redirected to malicious cryptocurrency addresses, potentially resulting in financial loss and undermining user trust in the application.

Potential Remediation:

- **Update Allowlist:** Regularly review and update the allowlist of approved redirect URLs to ensure it only includes trusted addresses.
- **Input Validation:** Implement strict validation on redirect URLs to prevent unauthorized redirects.
- **User Education:** Inform users about the risks of clicking on unexpected links or QR codes.

Vulnerable Components via Outdated Allowlist

RISK LEVEL: HIGH

Vulnerability Detail:
The Juice Shop application contains a suspicious library entry in its `3rdpartylicenses.txt`, specifically a typo in the library name “anuglar2-qrcode,” which should be “angular2-qrcode.” This raises concerns about potential typosquatting.

Discussion (Executive Summary):
The vulnerability was identified by accessing the `3rdpartylicenses.txt` file at `http://localhost:3000/3rdpartylicenses.txt` to review third-party libraries. Each library was verified against the npm repository, revealing the typo that could lead to the use of a malicious package.

Evidence of Validation:
Searching for “anuglar2-qrcode” on npm confirmed that it is a potentially typosquatted package, designed to deceive users.

Probability of Exploit/Attack:
The likelihood of exploitation is medium, as users may inadvertently install a malicious package due to the typo.

Impact:
If exploited, this vulnerability could lead to the introduction of malicious code into the application, compromising its security and integrity.

Potential Remediation:

- **Correct Typo:** Update the library name in the `3rdpartylicenses.txt` file to reflect the correct package name.
- **Review Dependencies:** Conduct a thorough review of all dependencies to ensure they are legitimate and secure.
- **Implement Security Practices:** Regularly audit third-party libraries for known vulnerabilities.

Unsigned JWT Vulnerability

RISK LEVEL: HIGH

Vulnerability Detail:
The Juice Shop application does not properly validate JWT signatures, allowing attackers to manipulate tokens and gain unauthorized access.

Discussion (Executive Summary):
This vulnerability was identified by capturing a valid JWT from the browser's cookies and manipulating it using JWT.io. The algorithm was changed from RS256 to "none," allowing the application to accept the token without verification.

Evidence of Validation:
The manipulated JWT was successfully replaced in the browser's cookie storage, allowing access to protected resources.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can easily forge JWTs without proper validation.

Impact:
If exploited, this vulnerability could allow unauthorized access to user accounts and sensitive data, leading to potential data breaches.

Potential Remediation:

- **Implement Signature Validation:** Ensure that the server validates JWT signatures properly and does not accept tokens with the "none" algorithm.
- **Use Strong Signing Algorithms:** Enforce the use of strong signing algorithms like RS256 for JWTs.
- **Regular Security Audits:** Conduct regular audits of authentication mechanisms to identify and mitigate vulnerabilities.

Local File Read (LFI) Vulnerability

RISK LEVEL: HIGH

Vulnerability Detail:
The Juice Shop application is vulnerable to Local File Inclusion (LFI) through the `/dataerasure` endpoint, allowing attackers to read sensitive files from the server.

Discussion (Executive Summary):
This vulnerability was identified by sending a test request to the `/dataerasure` endpoint and intercepting the request using Burp Suite. Fuzzing the parameters revealed that the server attempts to access files based on the `layout` parameter.

Evidence of Validation:
An invalid value for the `layout` parameter resulted in a `500 Internal Server Error`, indicating that the server tried to access a non-existent file.

Probability of Exploit/Attack:
The likelihood of exploitation is high, as attackers can manipulate the `layout` parameter to read sensitive files.

Impact:
If exploited, this vulnerability could lead to unauthorized access to sensitive files, including configuration files and user data.

Potential Remediation:

- **Input Validation:** Implement strict validation on the `layout` parameter to prevent directory traversal.
- **Restrict File Access:** Limit the files that can be accessed through the application to only those necessary for functionality.
- **Conduct Security Reviews:** Regularly review code and configurations to identify and mitigate LFI vulnerabilities.

Supply Chain Attack Vulnerability

RISK LEVEL: MEDIUM

Vulnerability Detail:
The Juice Shop application may be susceptible to supply chain attacks due to potentially vulnerable packages listed in its `package.json`.

Discussion (Executive Summary):
This vulnerability was identified by reviewing the `package.json` file for dependencies and development dependencies. A thorough review of each package was conducted to identify any known vulnerabilities.

Evidence of Validation:
Known vulnerabilities associated with certain packages were researched, revealing potential risks.

Probability of Exploit/Attack:
The likelihood of exploitation is medium, as attackers may exploit vulnerabilities in overlooked development tools.

Impact:
If exploited, this vulnerability could lead to the introduction of malicious code into the application, compromising its security.

Potential Remediation:

- **Regular Dependency Audits:** Conduct regular audits of all dependencies to identify and address known vulnerabilities.
- **Update Packages:** Ensure that all packages are up-to-date and sourced from reputable repositories.
- **Implement Security Best Practices:** Follow best practices for dependency management to mitigate supply chain risks.

Methodology

Assessment Tools Selection

The following tools were utilized during the assessment:

- **Web Browser:** For navigating the Juice Shop application and capturing requests.
- **Burp Suite:** For intercepting and analyzing HTTP requests and responses.
- **GoBuster:** For discovering hidden directories and files.
- **JWT.io:** For decoding and manipulating JSON Web Tokens.
- **Fuzzer:** For testing parameters and identifying vulnerabilities.

Assessment Methodology Detail

The assessment methodology involved several key steps:

1. Initial Analysis:

- Review of the application’s functionality and identification of potential attack vectors.
- Examination of third-party libraries and their licenses to identify any suspicious components.

2. Testing for Vulnerabilities:

- **Unvalidated Redirects:** Manipulated redirect URLs to test for outdated allowlist vulnerabilities.
- **JWT Manipulation:** Captured and modified JWTs to test for signature validation issues.
- **Local File Inclusion (LFI):** Fuzzed parameters in the `/dataerasure` endpoint to identify file access vulnerabilities.
- **Supply Chain Attacks:** Reviewed dependencies in `package.json` for known vulnerabilities.

3. Documentation of Findings:

- Each identified vulnerability was documented with details on the risk level, impact, and potential remediation strategies.

4. Recommendations:

- Provided actionable recommendations for mitigating identified vulnerabilities and improving the overall security posture of the application.