OWASP Juice Shop
Penetration Test Report

Version	**1.0**
Author	**Rana Emad 43-2101** **Ahmed Atef Askora 43-18023**
Issue Date	**08/06/2022**

Table of Contents

1.0 OWASP Juice Shop Penetration Test Report

1.1 Introduction

Subject of this document is a summary of penetration tests performed against web applications owned by OWASP Juice Shop company. Test was conducted according to rules of engagement defined and approved at the beginning by both parties – customer and contractor. Black-box pentesting assignment was requested.

Black-box penetration test classification means that the penetration tester has no internal knowledge about target system architecture. He/she can use information commonly available on the Internet. More data can be collected during the reconnaissance phase based on observation of target system behavior. Black-box penetration test results give overview of vulnerabilities exploitable from outside the company network. It shows how unauthenticated actors can take advantage of weaknesses existing in tested web applications.

Time frame defined:

Penetration test start: 06/05/2022,

Penetration test end: 08/06/2022.

1.2 Scope

To perform a Black Box Web Application Penetration Test against the web applications of the organization named OWASP Juice Shop.

This is what the client organization defined as scope of the tests:

- Dedicated Web Server: http://localhost:3000/#/
- Domain: http://localhost:3000/#/
- Subdomains: none.

2.0 – EXECUTIVE SUMMARY

The conducted penetration test uncovered several security weaknesses present in web applications owned by the OWASP Juice Shop company. When performing the penetration test, there were several alarming vulnerabilities that were identified on OWASP Juice Shop networks. When performing the attacks, we were able to gain access to multiple vulnerabilities, primarily due to outdated patches and poor security configurations. During the testing, the vulnerabilities found ranged between high vulnerabilities, that is assigned the highest priority, and low vulnerabilities with the following counts:

- 10 High, 6 Med, 4 Low issues have been identified.

2.1 – Recommendations

We recommend patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered at a later date.

3.0 – RISK ASSESSMENT

3.1 Likelihood

The likelihood is a measurement of the capacity to carry out an attack. The factor will be the difficulty or skill required to carry out the attack.

Risk	Description
Critical	An attacker is near-certain to carry out the threat event
High	An untrained user could exploit the vulnerability. The vulnerability is obvious or easil accessed
Medium	The vulnerability required some hacking knowledge to carry out the attack
Low	The vulnerability required significant time, skill , access and other resources to carry out th attack

3.2 Impact

The impact is a measurement of the adverse effect carrying out an attack would have on the organization.

Risk	Description
Critical	An attack would cause catastrophic or severe effect on operation, asset or other organization
High	An attack would severely degrade mission capability. The attack may result in damage to asse data exposure)
Medium	An attack would degrade the mission capability. An attack would allow for primary function t application to resume, but at reduced effectiveness
Low	An attack would degrade mission capability in a limited capacity. The attack may result i marginal damage to assets

4.0 -- FINDINGS SUMMARY

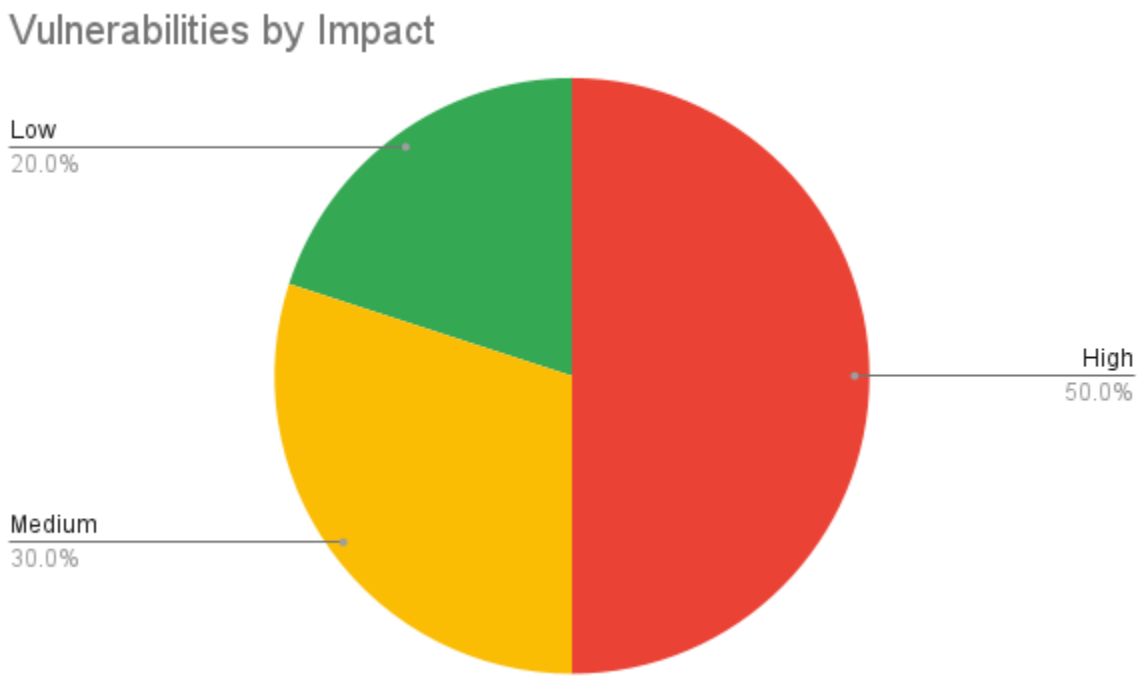
Findings	Likelihood	Impact	Rating	Status
Confidential Document (Sensitive Data Exposure)	Medium	Critical	High	**Open**
Login with Bender's account (SQL Injection)	Medium	Critical	High	**Open**
API-Only XSS	Medium	Critical	High	**Open**
Forged Review (Broken Access Control)	High	High	High	**Open**
Guessing Admin Credentials (Broken Authentication)	High	High	High	**Open**
Login Admin (SQL Injection)	High	High	High	**Open**
Admin Registration (Improper Input Validation)	Medium	High	High	**Open**
Christmas Special (SQL Injection)	Medium	High	High	**Open**
View another user's shopping basket (Broken Access Control)	Medium	High	High	**Open**
Deluxe Fraud (Improper Input Validation)	Medium	High	High	**Open**
Admin Section (Broken Access Control)	High	Medium	Medium	**Open**
DOM XSS Attack	High	Medium	Medium	**Open**
Bonus Payload (XSS)	High	Medium	Medium	**Open**
Reflected XSS Attack	Medium	Medium	Medium	**Open**
Receive a coupon from the chatbot	Medium	Medium	Medium	**Open**
CSRF Username Changing	Medium	Medium	Medium	**Open**
Forged Feedback (Broken Access Control)	Medium	Low	Low	**Open**
Repetitive Registration (Improper Input Validation)	Low	Low	Low	**Open**
Error Handling (Security Misconfiguration)	Low	Low	Low	**Open**
Give a Feedback without rating (Improper Input Validation)	Low	Low	Low	**Open**

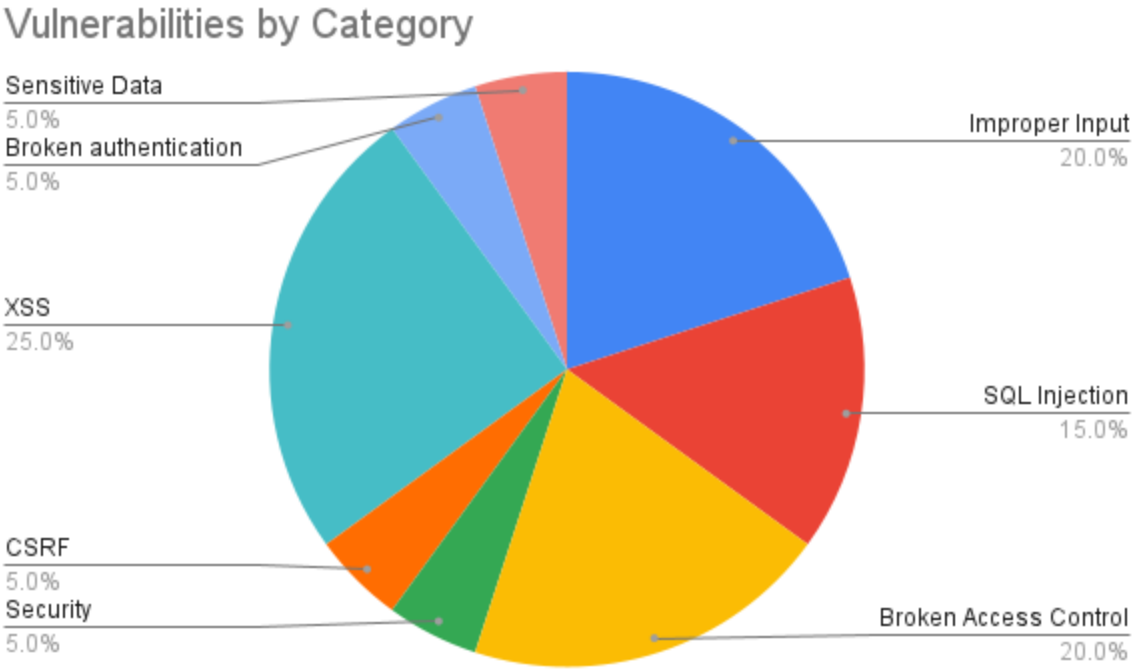
5.0 – VULNERABILITY & REMEDIATION REPORT

Penetration test finding classification, description and recommendations mentioned in the report are taken mostly from OWASP TOP 10 project documentation available on site: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

The OWASP TOP 10 is a list of definitions of web application vulnerabilities that pose the most significant security risks to organizations when exploited.

Vulnerability Summary




Vulnerabilities by Category
Sensitive Data
5.0%
Broken authentication
5.0%
XSS
25.0%
CSRF
5.0%
Security
5.0%
Improper Input
20.0%
SQL Injection
15.0%
Broken Access Control
20.0%

6.0 – INFORMATION GATHERING

The information gathering portion of a penetration test focuses on identifying the scope of the penetration test. We started the pentest with finding and discovering the endpoints and hidden features.

We have used multiple tools to aid us in achieving our target.

Tools Used:

1) Burp Suite
2) Postman
3) DevTools

7.0 High Findings

7.1.Title: Confidential Document

Rating: High

URL: http://localhost:3000/ftp/acquisitions.md

Description: Access a confidential document

Proof of Concept:

Step 1: Open the side menu and go to the About Us page then follow the link titled “Check out our boring terms of use if you are interested in such lame stuff”.

Step 2: Changing the URL into http://localhost:3000/ftp to browse the directory.

Step 3: Open http://localhost:3000/ftp/acquisitions.md to access a confidential document.

Remediation Steps:

-Avoid supplying the file system API with user input altogether.

-Validate the input to ensure only alphanumeric characters are supplied.

-Use a platform API to canonicalize the path and make sure it starts with the expected base directory.

7.2.Title: Login Admin

Rating: High

URL: https://juice-shop-rana.herokuapp.com/#/login

Description: Log in with the administrator’s user account.

Proof of Concept:

Step 1: In the email textbox on the login page, use the basic ‘ OR 1=1 - - SQL injection command.

Step 2: Write any text in the password textbox and log in.

Step 3: The system logs in with the first entry in the users table.

Remediation Steps:

-Using white-lists based validation.

7.3.Title: Login with another user account

Rating: High

URL: http://localhost:3000/#/login

Description: Login with Bender’s account

Proof of Concept:

Step 1: In the email textbox on the login page, Enter ' OR email like ('%bender%'); -- SQL injection command.

Step 2: Write any text in the password textbox and log in.

Step 3: The system logs in with Bender's account.

Remediation Steps:

-Using white-lists based validation.

7.4.Title: Christmas Special

Rating: High

URL: http://localhost:3000/rest/products/search?q=

Description: Order the Christmas special offer of 2014

Proof of Concept:

Step 1: The vulnerability of the provided url is that it returns the products, so SQL injection on that endpoint using the query " q='))-- " retrieves all products.

Step 2: We search for "christmas" to find the christmas offer and get the ProductId

Step 3: Using the Network tab in chrome's tools, we identify the payload and the url when adding an item to the basket

Step 4: Sending a POST request to the url http://localhost:3000/api/BasketItems with payload { "BasketId" : "6" , "ProductId" : 10, "quantity" : 1} where the basketId can be observed in the BasketItems request in the Network tab.

Step 5: Go to your basket and checkout

Remediation Steps:

Since the main vulnerability is exposed using SQL injection, white-lists based validation should be used.

The deleted products should be stored in a separate endpoint from the active products.

The basket api should not be accessible by the client-side. Allow only internal requests.

7.5.Title: Admin Registration

Rating: High

URL: http://localhost:3000/api/Users

Description: Register as a user with administrator privileges

Proof of Concept:

Step 1: Exposing the users api by intercepting the register functionality, the vulnerability is exploited by sending a POST request to http://localhost:3000/api/Users with the normal login payload {“email” : “admin”, “password”: “admin”, … , “role”: “admin”}

Step 2: the software does not validate the input properly therefore, adding the role was possible, gaining the attacker admin privilege.

Remediation Steps:

Adding or adjusting the role must not be possible for the registering endpoint (client side). The payload as well as the source url shall both be validated, rejecting admin-related requests coming from customer related endpoints.

7.6.Title: Guessing Admin Credentials

Rating: High

URL: http://localhost:3000/#/login

Description: Try to login as an admin by guessing weak passwords

Proof of Concept:

Step 1: Visit http://localhost:3000/#/login.

Step 2: Login with Email admin@juice-sh.op and Password admin123 which is as easy to guess.

Remediation Steps:

Use Strong password.

7.7. Title: View another user's shopping basket

Rating: High

URL: http://localhost:3000/#/basket

Description : View another user's shopping basket

Proof of Concept:

Step 1 :View your basket by clicking on Your Basket button in the app bar.

Step 2 : Inspect the Session Storage in your browser's developer tools to find a numeric bid value.

Step 3 : Change the bid value and reload the page.

Remediation Steps:

-Ensure Lookup IDs are Not Accessible Even When Guessed or Cannot Be Tampered With

7.8. Title: Forged Review

Rating: High

URL: http://localhost:3000/#/

Description : Post a product review as another user or edit any user's existing review

Proof of Concept:

Step 1 :Select any product and write a review for it

Step 2 : Submit the review and intercept the request using Burp Suite Proxy

Step 3: Manipulate the author parameter in the payload. The payload will look like:

{ “message”: “Here is the message”, “author” : “admin@juice-sh.op”}

Remediation Steps:

- Since posting the feedback did not rely on the logged in user, allow only authenticated logged in users to post reviews.
- The author parameter shall be removed from the body payload, and instead it should be fetched from the user session.

7.9. Title: Deluxe Fraud

Rating: High

URL: http://localhost:3000/#/payment/deluxe

Description : Obtain a Deluxe Membership without paying for it

Proof of Concept:

Step 1 : Inspecting the upgrade to deluxe page using the dev tools, the pay button next to “pay using wallet” is disabled, since there’s not enough funds in the wallet.

Step 2 : Remove the disabled= “true” and the class mat-button-disabled from element. The button is now enabled.

Step 3: Click on pay and intercept the payment process using burp suite proxy. The payload is set to {“paymentMode” : “wallet”}.

Step 4: change the payment mode to be empty: {“paymentMode” : “”}, upgrading the user to deluxe.

Remediation Steps:

-Better input validation from the server-side is required. The upgrade should not be effective until the payment is processed successfully.

7.10. Title: API-Only XSS

Rating: High

URL: http://localhost:3000/api/Products

Description : Perform persistent XSS attack using POST request to the Products api, adding a new product and injecting XSS script in its description.

Proof of Concept:

Step 1 : The attacker creates a POST request to the Products API. The normal request payload includes the product name, description and price. The script is injected in the description to look like this:

{“name” : “XSS” , “description” : “ <iframe src=\”javascript:alert(‘xss’)\”>”, “price” : 20.33}

Step 2 : On visiting the search endpoint, displaying the products, the new product created by the attacker is loaded along with the other products, loading the script from the description. An alert box with “xss” appears. The product is created and injected with the other products, creating a very dangerous persistent xss attack.

Remediation Steps:

-The product api endpoint should only be accessible to admins as they are the only users that should be allowed to post new products. Therefore, cookies need to be implemented checking the current logged in user is an admin user.

-White-list validation should be implemented in the description field to ensure that it can not be injected with dangerous code such as script. Since the description field does not need special characters like “<”, it shall be possible and better to restrict the input to alphanumeric characters only.

8.0 Medium Findings

8.1. Title: DOM XSS Attack

Rating: Medium

URL: http://localhost:3000/#/search?q=

Description: An attacker can perform a DOM XSS attack which can inject HTML and run JavaScript code on the client side.

Proof of Concept:

Step 1: Enter The attack string <iframe src="javascript:alert(`xss`)"> into the Search field.

Step 2: An alert box with the text "xss" should appear.

Remediation Steps:

- Input validation should be applied, including encoding, white-list validation, removal of dangerous characters such as <>

- Context aware output encoding in case the attacker was able to bypass the input validation.

8.2. Title: Bonus Payload

Rating: Medium

URL: http://localhost:3000/#/search?q=

Description: Same as the DOM XSS attack, the attacker can inject a script into the search field.

Proof of Concept:

Step 1: Instead of using the alert method, displaying an alert box, the src is injected with a link to a soundtrack on Soundcloud using the payload: <iframe allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>

Step 2: The soundtrack is fetched and played.

Remediation Steps:

- Input validation should be applied, including encoding, white-list validation, removal of dangerous characters such as <>

- Context aware output encoding in case the attacker was able to bypass the input validation.

8.3. Title: Reflected XSS Attack

Rating: Medium

URL: http://localhost:3000/#/track-result?id=

Description: Perform a reflected XSS attack

Proof of Concept:

Step 1:Visit the Order History then click on the truck button.

Step 2: Notice that the URL has changed to http://localhost:3000/#/track-result?id=SOME_ID

Step3: Replace the id with the reflected XSS attack string <iframe src="javascript:alert(`xss`)">.

Remediation Steps:

- Input validation.
- Context aware output encoding

8.4.Title: Admin Section

Rating: Medium

URL: http://localhost:3000/#/administration

Description: Access the administration section of the store

Proof of Concept:

Step 1: Exploring the browser's developers tools, in the Sources tab, scan the main.js file

Step 2: One of the paths that can be routed to, is the administration path

Step 3: While logged in as admin, entering http://localhost:3000/#/administration in the url, routes to the administration section. If you are logged in with a normal customer account, the url gives a 403 forbidden error.

Remediation Steps:

-Enforce authorization steps when accessing sensitive routes / paths that require high privilege users.

-Sensitive paths should not be in a guessable place or have descriptive guessable names such as admin, as this is the first search keyword that comes to mind to access admin related sensitive data.

8.5. Title: Receive a coupon code from the chatbot.

Rating: Medium

URL: http://http://localhost:3000/#/chatbot

Description: Receive a discount coupon code from the chatbot.

Proof of Concept:

Step 1: Open the side menu and go to Support Chat

Step 2: Ask the bot something similar to "Can I have a coupon code?" or "Please give me a discount!" and it will most likely decline with some unlikely excuse.

Step3: Keep asking for a discount again and again until you finally receive a 10% coupon code.

Remediation Steps:

-Keep rejecting the client's question.

8.6. Title: CSRF Username Changing

Rating: Medium

URL: http://localhost:3000/profile

Description: Change the username of another user by performing Cross-Site Request Forgery from another Origin

Proof of Concept:

Step 1: Navigate to the profile page after logging in.

Step 2: Change your username and intercept the request using Burp Suite. The cookies is used to identify the user.

Step 3: An HTML page can be created and hosted on the attacker's exploit server. The HTML page would look like this:

<form action= "http://localhost:3000/profile" method= "POST">
    <input name= "username" value= "target username"/>
    <input type= "submit" />
</form>
<script> document.forms[0].submit(); </script>

Step 4: The target's username will be changed once the victim visits the attacker's exploit page.

Remediation Steps:

To mitigate this CSRF attack, the website needs to verify the identity of the source of the request using anti-CSRF token, where the token is checked to ensure it is linked to the user in the user session. If the token is not present or has expired, the website should be blocked.

9.0 Low Findings

9.1. Title: Repetitive Registration

Rating: Low

URL: http://localhost:3000/#/register.

Description : Register with improper password validation

Proof of Concept:

Step 1 : Go to http://localhost:3000/#/register and fill out all the required inputs.

Step 2 : After filling out the Password field that matches the Repeat Password field, go back to the password field and change it.T he Repeat Password field does not show the expected error.

Step 3 : Submit the form with Register will be accepted although the two password does not match.

Remediation Steps:

Make sure that the value of the two fields are identical while changing any of the two fields instead of checking the order of validation.

9.2. Title: Error Handling

Rating: Low

URL: https://juice-shop-rana.herokuapp.com/rest/blah

Description : Provoke an error that is neither very gracefully nor consistently handled

Proof of Concept:

Step 1 : At the endpoint /rest/ adding any text that is not meaningful to the system such as “blah”, redirects to the 500 error: unexpected path

Step 2 : The 500 error exposes valuable information about the system that should be concealed.

Remediation Steps:

Issues at error handling can reveal a lot of information about the system which may be used to identify new injection points or system vulnerabilities. This can be avoided by using generic error messages to redirect the user and not give away information about the system.

9.3. Title: Give a Feedback without rating

Rating: Low

URL: http://localhost:3000/#/contact

Description : Give a feedback to the store with zero star although rating field is required

Proof of Concept:

Step 1 : Open the side menu and select customer feedback.

Step 2 : Fill out the required fields except the rating field.

Step 3 : Inspect the Submit button with DevTools and note the disabled attribute of the <button> HTML tag.

Step 4: Remove the disabled attribute and Submit the form and you can check the zero star rating in the About Us page.

Remediation Steps:

Check that all required fields are fulfilled before submitting the form, not just disable the button attribute.

9.4. Title: Forged Feedback

Rating: Low

URL: http://localhost:3000/#/contact

Description : Posting feedback in another user's name

Proof of Concept:

Step 1 : In the Contact Us page, fill the form.

Step 2 : Submit the form and intercept it using Burp Suite Proxy. The payload sent is {"captchaId": 5, "captcha": "24", "comment": "New Comment heree (anonymous)" , "rating" :2}

Step 3 : Add the UserId of the intended user as a parameter in the payload to look like: {"UserId": "2", "captchaId":5, "captcha": "24", "comment": "New Comment heree (anonymous)" , "rating" :2}, then forward the request and the feedback will be submitted successfully.

Remediation Steps:

This vulnerability could be mitigated by avoiding using the UserId as input in the payload, and getting the currently active user from the user sessions.