Juice Shop
Penetration Test Report

Version	1.0
Author	Karim amr, hamza shafik, Youssef amr
Issue Date	15/12/2024

Table of Contents

1.0 Juice Shop Penetration Test Report

1.1 Introduction

Subject of this document is a summary of penetration tests performed against web applications owned by 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 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: 8/12/2024,

Penetration test end: 14/12/2024.

1.2 Scope

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

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

- ☒ Dedicated Web Server: http://localhost:3000/#
- ☒ Domain: domain.com
- ☒ Subdomains: all subdomains

2.0 – EXECUTIVE SUMMARY

Conducted penetration test uncovered several security weaknesses present in web applications owned by Juice Shop company

When performing the penetration test, there were several alarming vulnerabilities that were identified Juice Shop networks. When performing the attacks, I was able to gain access to multiple sensitive areas, primarily due to outdated patches and poor security configurations. During the testing, I had unauthorized access to administrative portal. These systems as well as a brief description:

- Subdomain Found: None, All identified endpoints are within the main domain

- 15 High, 8 Med, 7 Low issue has been identified.

2.1 – Recommendations

I 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 attach.

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 easily accessed
Medium	The vulnerability required some hacking knowledge to carry out the attack
Low	The vulnerability required significant time, skill , access and other resource to carry out the 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 asset(data exposure)
Medium	An attack would degrade the mission capability. An attack would allow for primary function to application to resume, but at reduced effectiveness
Low	An attack would degrade mission capability in a limited capacity. The attack may result in marginal damage to assets

4.0 -- FINDINGS SUMMARY

Findings	**Likelihood**	**Impact**	**Rating**	**Status**
Repetitive Password	Medium	Critical	High	**Open**
Loging in as a user (sql injection)	Medium	Critical	High	**Open**
API only XSS	High	High	High	**Open**
Client Side XSS Protection	Medium	Critical	High	**Open**
Captcha Bypass	High	High	High	**Open**
Bypass Upload Size Restriction	Medium	High	High	**Open**
GDPR Data Theft	Medium	Medium	Medium	**Open**
Reset Another User's Password(bruteforce)	Medium	Medium	Medium	**Open**
Bjorn's Security Question	High	Medium	Medium	**Open**
Accessing Developer's Backup Files	Medium	High	High	**Open**
Manipulate Rating	Low	Low	Low	**Open**
Access Confidential Documents	Low	Low	Low	**Open**
Dom XSS	Low	Low	Low	**Open**
Missing Encoding	Low	Low	Low	**Open**
Login mcsafesearch	Low	Low	Low	**Open**
Manipulate Basket	Medium	Medium	Medium	**Open**
Privacy Policy Inspection	Low	Low	Low	**Open**
Change Bender's password	Medium	High	High	**Open**
Use Noncompatible Upload Type	Medium	Medium	Medium	**Open**
Empty User Registration	Low	Low	Low	**Open**
Unvalidated Redirect	Medium	Medium	Medium	**Open**

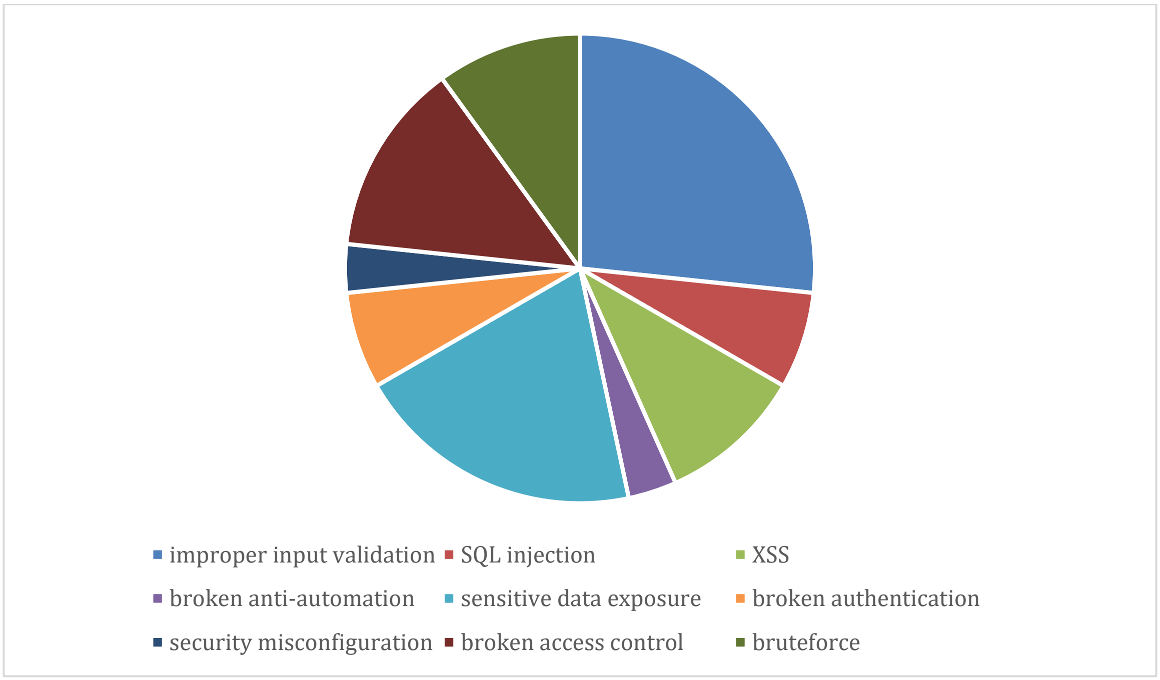
Login As Admin (SQL injection)	High	Critical	High	**Open**
Revealing Error	Medium	High	High	**Open**
Access Administration Page	Medium	High	High	**Open**
Login As Admin (Bruteforce attack)	High	Critical	High	**Open**
View Another User’s Basket	Medium	High	High	**Open**
Register As An admin	High	Critical	High	**Open**
Forged Feedback	Medium	High	High	**Open**
Change Account Balance	Medium	Medium	Medium	**Open**
Get Free Coupons	Medium	Medium	Medium	**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



6.0 – INFORMATION GATHERING

The information gathering portion of a penetration test focuses on identifying the scope of the penetration test. I started the pentest with finding all subdomains.

I have used multiple tools to make sure that I haven’t missed any domain.

Tools Used:

1) WFUZZ
2) Virustotal
3) Sublist3r

7.0 High Findings

7.1 Title: Repetitive Password

Rating: High

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

Description:

While registering a new user, if “Repeat Password” input field doesn’t not match the “Password” input field it displays “Passwords do not match”. But if they initially match, you can then change the “Password” to anything else

Proof of Concept:

Step 1: go to the registration page

Step 2: type a password in the Password field

Step 3: type the same password in the Repeat Password field

Step 4: go back again into the password field and write any other password(may be different than the one we used in the Repeat Password field)

Step 5: click Register and you will register successfully

Remediation Steps:

1)Implement Server-Side Validation:

Ensure that the server verifies the Password and Repeat Password fields match before processing the registration request.

Reject requests where the Password and Repeat Password fields do not align.

2) Disable Client-Side Validation Bypassing:

Prevent users from editing form inputs after the initial validation by disabling or locking the Password field once the Repeat Password field is filled.

7.2 Title: Logging in as a different user

Rating: High

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

Description:

- SQL Injection
- When a user posts a feedback on a product it states his email and feedback description, so we took bender's email and used it login

Proof of Concept:

Step 1: view a product and get target's email

Step 2: go to the login page

Step 3: in the email input field type the email followed by – (ex: bender@juice-sh.op). the double dash comments anything coming after the email attribute

Step 4: press login and you will be successfully logged in as bender

Remediation Steps:

1)**Use Parameterized Queries:**

Replace dynamic SQL queries with parameterized or prepared statements to prevent the injection of malicious SQL code.

2)**Sanitize User Inputs:**

Validate and sanitize all inputs submitted by the user.

Reject or escape characters that have special meaning in SQL (e.g., --, ', ;).

7.3 Title: API only XSS

Rating: **High**

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

Description:

perform a persisted XSS attack with an iframe without using the frontend

Proof of Concept:

Step 1: login and view products

Step 2: switch to burp and intercept the get /api/Products packet and send it to repeater

Step 3: change the get request into a put request

Step 4: take the data(attributes) received from the response and add it to the new packet we are making and instead write(<iframe src=\”javascript:alert(‘alert’)\”>)

Step 5: get an admin’s authorization token and add it instead of the one in the old packet

Step 6: now if anyone views the product we modified an alert will pop up

Remediation Steps:

Sanitize and Validate Input on the Server Side:

- Reject any malicious or unexpected input sent to the API endpoints.
- Sanitize input data to escape or remove potentially harmful tags, such as <script> or <iframe>, before storing or processing it.

7.4 Title: Client Side XSS Protection

Rating: High

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

Description:

perform a persisted XSS attack with an iframe bypassing a client-side security mechanism

Proof of Concept:

Step 1: go to the registeration page

Step 2: register a new user

Step 3: switch to burp and intercept the post request

Step 4: write (<iframe src=\"javascript:alert('alert')\">) instead of the email

Step 5: now whenever an admin views this user an alert will pop up

Remediation Steps:

Implement Proper Output Encoding:

Encode all user-supplied content when rendering it in the frontend. For instance:

- Use HTML encoding to escape special characters.
- Example: <iframe> should be rendered as &lt;iframe&gt;.

7.5 Title: Captcha Bypass

Rating: High

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

Description:

Send the same captcha in many requests

Proof of Concept:

Step 1: go to the customer feedback page and post a feedback

Step 2: switch to burp and send the post request to the intruder

Step 3: generate 10 null payloads and send them

Remediation Steps:

Implement Unique Captcha Tokens per Session:

- Generate a unique Captcha token for each session or user interaction.
- Ensure that the Captcha token is invalidated immediately after a successful submission.

7.6 Title: Bypass Upload Size Restriction

Rating: High

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

Description:

The maximum file upload size is supposed to be 100kb , in this vulnerability I uploaded a file larger than 100kb

Proof of Concept:

Step 1: go to the complaint page and choose a file and send complaint

Step 2: switch to burp and send the post request to the repeater

Step 3: open the file you want to upload in notepad and copy the data

Step 4: change the old packet's data with the new data of the large file you want to upload

Remediation Steps:

Enforce File Size Limits on the Server Side:

- Validate the size of uploaded files on the server before processing them.
- Reject any file exceeding the allowed size limit with an appropriate error message.

Implement Hash-Based Integrity Check:

- Use hash functions to verify the integrity and size of the file content during the upload process.

7.7 Title: Accessing developer's backup files

Rating: High

URL: http://localhost:3000/ftp

Description:

Access hidden backup folders

Proof of Concept:

Step 1: use path /ftp

Step 2: open package.json.bak , we will find that only .md and .pdf files are allowed

Step 3: null byte inject the path

Step 4: send the injected path to burp encoder and encode it as url

Remediation Steps:

Restrict Access to Sensitive Directories:

Prevent public access to directories like /ftp by modifying the server configuration to block access or make them inaccessible via the web

7.8 Title: Change Bender's Password

Rating: High

URL: http://localhost:3000/privacy-security/change-password

Description:

Change a user's password by manipulating the change password form

Proof of Concept:

Step 1: login into bender's account (already exploited)

Step 2: go to the change password form

Step 3: switch to burp and send the get request to the repeater

Step 4: remove the current attribute in the api and leave new and repeat

Step 5: after sending the request we will find that bender's password is now changed

Remediation Steps:

Validate Current Password on the Server Side:

- Require the currentPassword field in the request payload and validate it against the stored password hash in the database.
- Reject requests missing this attribute or containing an invalid current password.

7.9 Title: Login as Admin

Rating: High

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

Description:

Login with the administrator's user account using SQL injection

Proof of Concept:

Step 1: go to the login page

Step 2: in the email input form type: admin ' or 1=1

The quote used to tell sql this ends the sql entry and the double dash is to comment anything that comes after the username

Step 3: now you are logged into the admin's account

Remediation Steps:

Use Parameterized Queries:

Replace dynamic SQL queries with parameterized or prepared statements to prevent the execution of malicious SQL code.

Sanitize User Inputs:

- Reject or escape special characters such as ', --, ;, and "

7.10 Title: Revealing error

Rating: **High**

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

Description:

Provoked an error that reveals database software, error trace stack, absolute path of affected file including line number

Proof of Concept:

Step 1: go to the login page

Step 2: in the email input form type: ‘ and any password

Step 3: switch to burp and view the response packet

Remediation Steps:

Disable Detailed Error Messages in Production:

- Ensure that detailed error messages, including stack traces, database information, and file paths, are not exposed in the production environment.

Configure Proper Error Handling:

- Implement proper error-handling mechanisms for different types of errors (e.g., validation errors, database connection errors).

7.11 Title: Access Administration Page

Rating: **High**

URL: http://localhost:3000/administration

Description:

Access a hidden administration page and can manage users and feedbacks

Proof of Concept:

Step 1: inspect the main javascript file in the sources , found : path= /administration

Step 2: login as admin(already exploited)

Step 3: go to the path /administration

Remediation Steps:

Check User Role on the Server Side (Not Client-Side):

- Never rely on client-side code (e.g., JavaScript) to control access to sensitive areas, as it can be easily bypassed. All access control must be enforced on the server side.
- Ensure that sensitive paths like /administration are only accessible if the server-side application verifies the user's role and authorization.

7.12 Title: Login as Admin

Rating: **High**

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

Description:

We already know the admin’s email is: admin@juice-sh.op so we will use bruteforce attack to get the admin’s password

Proof of Concept:

Step 1: login with email : admin@juice-sh.op and any password

Step 2: switch to burp and send packet to intruder

Step 3: select the password attribute to be the attribute that will be checked in the attack

Step 4: start attack , after the attack is finished you will see the admin’s password

Remediation Steps:

Implement Account Lockout Mechanism:

- Implement an account lockout mechanism that temporarily disables the login attempt after a certain number of failed login attempts. This will prevent brute force attacks.

7.13 Title: View Another User’s Basket

Rating: High

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

Description:

View another user’s basket without having permission

Proof of Concept:

Step 1: view your basket

Step 2: switch to burp to view the get packet and send to repeater

Step 3: we see in the request: /rest/Basket/6

So we know that 6 is our basketed. if we change it we will see other user’s basket

Remediation Steps:

User Authorization:

- Implement authorization mechanisms that verify the relationship between the user and the requested resource (e.g., basket). Access should be restricted based on the ownership of the resource.

Input Validation:

- Validate and sanitize all input to prevent tampering with identifiers (e.g., basket IDs). Ensure that requests are scoped to the authenticated user's data.

7.14 Title: Register as an Admin

Rating: High

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

Description:

Register a new user with admin privileges

Proof of Concept:

Step 1: register as a new normal user

Step 2: switch to burp to view the post packet and send to repeater

Step 3: we see in the response there is a "role" attribute

Step 4: add role:admin to the new request and send

now this new user is registered as an admin

Remediation Steps:

Modify the Response to Exclude Sensitive Data:

Instruct the developer to omit sensitive attributes (like the role field) from the response after a successful user registration. The response should only include a confirmation message indicating that the user has been registered successfully.

7.15 Title: Forged Feedback

Rating: High

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

Description:

Post a feedback with another user's credentials

Proof of Concept:

Step 1: go to the customer feedback page and send a feedback

Step 2: switch to burp and view the post

we see that the response includes userid

Step 3: send the packet to the repeater and change the userid

Step 4: if we viewed the feedback now we will see the user that posted the feedback isn't you

Remediation Steps:

Use Strong Authentication:

Ensure that authentication (e.g., sessions or JWT token) is required for all feedback actions. This ensures that the user who submits feedback is the same as the one who is authenticated and logged in.

8.0 Medium Findings

8.1 Title: GDPR Data Theft

Rating: Medium

URL: http://localhost:3000/#/privacy-security/data-export

Description:

Steal someone else’s personal data

Proof of Concept:

Step 1: login and make an order

Step 2: switch to burp and view the request

We find that the vowels in the email have been escaped to *

Step 3: make a new account called edmin@juice0sh.op

Step 4: login and go to privacy security > request data export

Now I will receive the admin’s past orders

Remediation Steps:

Escape Characters Securely:

Instead of replacing only vowels with *, consider applying a more comprehensive approach to escape personal data. The goal is to ensure that any personally identifiable information (PII) or sensitive data is obfuscated in a way that it can't be easily guessed or reverse-engineered.

8.2 Title: Reset Another User’s Password

Rating: **Medium**

URL: http://localhost:3000/#/privacy-security/change-password

Description:

Reset another user’s password using bruteforce attack

Proof of Concept:

Step 1: get a user’s email from the product reviews (example: jim)

Step 2: now I have to guess jim’s security question: “eldest sibling’s middle name” , so I will write anything random and send the request to burp

Step 3: send the request to intruder and use a list of most common names to know the security question’s answer

Remediation Steps:

Implement Account Lockout Mechanism:

Implement an account lockout mechanism that temporarily disables the login attempt after a certain number of failed login attempts. This will prevent brute force attacks.

8.3 Title: Bjorn’s Security Question

Rating: **Medium**

URL: http://localhost:3000/#/forgot-password

Description:

Bjorn Kimmich is the one who created Owasp Juice Shop. With some searching on the web, an online video was found in which he goes through juice shop and registers using his email: bjoern@owasp.org and his pet’s name: Zaya

Proof of Concept:

Step 1: go to the login page and click on “ forgot your password?”

Step 2: now you have the email and the security question. So you can change the password

Remediation Steps:

Use a Stronger Authentication Method:

Replace or augment the security question with more secure authentication methods, such as multi-factor authentication (MFA) or email verification. This way, even if a security question is compromised, the overall system remains secure.

8.4 Title: Manipulate Basket

Rating: Medium

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

Description:

Put an additional product into another user's shopping basket

Proof of Concept:

Step 1: login and add an item to the basket

Step 2: switch to burp and send the post request to the repeater

Step 3: modify basket id and product id

Remediation Steps:

Use Secure Session Management:

Always associate the user's session or authentication token with the basket they are interacting with. This way, each request will be checked to ensure that the basket being modified belongs to the authenticated user.

8.5 Title: Use Noncompatible Upload Type

Rating: Medium

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

Description:

Upload a jpeg file when only pdf and zip are compatible

Proof of Concept:

Step 1: upload a normal pdf

Step 2: switch to burp and send the post request to the repeater

Step 3: open the jpeg file I want to upload in notepad to get access to the content

Step 4: replace the pdf file content in the request with the jpeg file I want to upload

Remediation Steps:

Validate File Types Server-side:

Ensure that the file type validation is performed server-side, not just on the client. When handling file uploads, check the MIME type and file extension to confirm that the file matches the allowed types (e.g., PDF and ZIP). Only allow files of the correct MIME type and extension to be uploaded, and reject any mismatched types.

8.6 Title: Unvalidated Redirect

Rating: **Medium**

URL: http://localhost:3000/#

Description:

Found a redirect link in the main javascript file while inspecting the website that redirects to a crypto wallet page

Proof of Concept:

Step 1: inspect sources , main.js

Step 2: find redirect?

Step 3: click on the link and you will be redirected

Remediation Steps:

Whitelisting Allowed URLs:

Maintain a whitelist of approved and trusted domains or URLs that the application can redirect users to. Before performing a redirect, check if the URL is on this list. If the URL is not on the list, the system should reject the redirect.

8.7 Title: Change Account Balance

Rating: **Medium**

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

Description:

Change my account balance

Proof of Concept:

Step 1: add an item to my basket

Step 2: switch to burp and intercept the message

Step 3: change the quantity attribute to -1000

Step 4: proceed to checkout and when you see the wallet you will find the redeemed money

Remediation Steps:

Server-Side Quantity Validation:

Ensure that negative quantities cannot be submitted when placing an order. On the server, validate that the quantity of items being added to the basket is always a positive integer (greater than 0).

8.8 Title: Get Free Coupons

Rating: **Medium**

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

Description:

Send many requests to the chatbot about getting free coupons until he sends you one

Proof of Concept:

Step 1: go to the support chat page and ask for a coupon

Step 2: switch to burp and send the post request to the repeater

Step 3: bruteforce (many attempts) until you receive a coupon response

Remediation Steps:

Implement Rate Limiting:

Limit the number of requests a user can send to the chatbot in a given time period (e.g., 5 requests per minute). This will prevent attackers from bruteforce attacking the chatbot by sending many requests in quick succession to obtain a coupon.

9.0 Low Findings

9.1. Title: Manipulate Rating

Rating: **Low**

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

Description:

Give zero stars rating when the choices are from 1-5

Proof of Concept:

Step 1: go to the customer feedback page and send a normal feedback with any rating

Step 2: switch to burp and send the post request to the repeater

Step 3: change the rating to 0

Remediation Steps:

Validate Ratings:

Ensure that the rating values are validated on the server side before they are accepted. Ratings should be restricted to the defined range (e.g., 1-5 stars) and any invalid values (e.g., 0 stars) should be rejected. If an invalid rating value is received, return an error message indicating that only ratings between 1 and 5 are allowed.

9.2 Title: Access Confidential Documents

Rating: **Low**

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

Description:

Access the acquisitions document

Proof of Concept:

Step 1: go to the about us page and press the green link button

You are then redirected to a page called legal.md

Step 2: switch to burp and send this request to the repeater

Step 3: replace legal.md with acquisitions.md

Remediation Steps:

Implement Access Control on Sensitive Resources:

- Ensure that access control checks are in place for sensitive documents or resources. If a document, such as acquisitions.md, should only be accessible to authorized users, implement proper role-based access control (RBAC) to restrict unauthorized users from accessing it. This will prevent users from manipulating the URL to access files they are not allowed to view.

Use Proper URL Parameterization:

- Avoid exposing the names of sensitive documents in the URL or path directly. Instead, use a more secure and abstract mechanism to reference sensitive documents, such as ID-based or token-based access control, where the document name or identifier is not easily guessed or modified.

9.3 Title: DOM XSS

Rating: Low

URL: http://localhost:3000/#

Description:

Inject a code snippet in the search bar and was able to alert the user

Proof of Concept:

Step 1: in the search bar type: <iframe src=”javascript:alert(‘xss’)”>

And an alert box will pop up

Remediation Steps:

Implement Proper Output Encoding:

Encode all user-supplied content when rendering it in the frontend. For instance:

- Use HTML encoding to escape special characters.
- Example: <iframe> should be rendered as &lt;iframe&gt;.

9.4 Title: Missing Encoding

Rating: Low

URL: http://localhost:3000/#/photo-wall

Description:

Retrieve the photo of Bjorn’s cat

Proof of Concept:

Step 1: inspect the webpage and find the url of the image

Step 2: encode # to the url which is equivalent to %23

Now the picture will appear

Remediation Steps:

Proper URL Encoding:

- Ensure that all user input or URL parameters are properly encoded before being included in the URL. Specifically, any special characters, such as #, should be percent-encoded (e.g., # should be encoded as %23). This ensures that URLs are interpreted correctly and that users cannot manipulate URLs to gain unintended access to resources.

9.5 Title: Login mcsafesearch

Rating: Low

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

Proof of Concept:

Step 1: login with the admin’s email(already exploited)

Step 2: switch to burp and send the message to the repeater

Step 3: change the email to mc.safesearch@juice-sh.op and the password to Mr.NOOdles

Remediation Steps:

Immediate Account Security Actions:

- Force a password reset for all affected accounts, especially those associated with the leaked information. Notify the impacted users about the incident and recommend creating strong, unique passwords.

Remove the Exposed Information:

- Identify where the sensitive information is published (e.g., forums, pastebins, public repositories).
- Request its removal by contacting the platform hosting the information. Use DMCA takedown requests or the platform's content removal policies, if applicable.

9.6 Title: Privacy Policy Inspection

Rating: **Low**

URL: http://localhost:3000/#/privacy-security/privacy-policy

Description:

Find the privacy policies of the website

Proof of Concept:

Step 1: open privacy policy page

Step 2: certain words will have an orange glow behind them when hovered upon. Inspect the page and you see there is .hot before the glowing url

Step 3: search for .hot and you will find all the keywords in the page

Remediation Steps:

Restrict Keyword Exposure:

- Avoid embedding sensitive or unnecessary metadata, such as specific CSS classes (.hot), that can expose structured information about the page content. Instead, dynamically generate or obfuscate such styling elements during runtime.

9.7 Title: Empty User Registration

Rating: **Low**

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

Description:

Register a new user with an empty email and password

Proof of Concept:

Step 1: register a new user

Step 2: switch to burp and send the post request to the repeater

Step 3: let the email and password have empty values and send the request

You will see that a new user with no email or password is created

Remediation Steps:

Input Validation on Client Side:

Ensure that the registration form has mandatory fields for email and password. Use HTML5 form validation attributes like required and pattern to enforce basic input checks at the client side.