Juice Shop

C4PO
v.0.0.1

OWASP

Penetration Test
Report of Findings

Cel

07/31/2023

Version 1.0

Table of Contents

Appendencies

Findings Severities

Risk Matrix

Severity Rating Definitions

State of Confidentiality

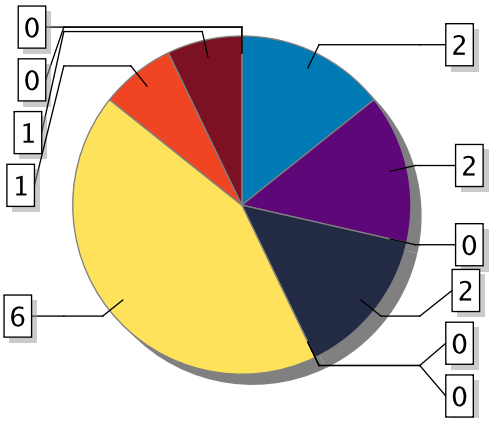
The contents of this document are considered to be proprietary and business confidential information. This information is to be used only in the performance of its intended use. This document may not be released to another vendor, business partner or contractor without prior written consent. Additionally, no portion of this document may be communicated, reproduced, copied or distributed without the prior consent. The contents of this document do not constitute legal advice. The offer of services that relate to compliance, litigation or other legal interests are not intended as legal counsel and should not be taken as such. The assessment detailed herein is against the company for examination purposes, and the vulnerabilities included in this document should be mitigated in order to secure external and / or internal infrastructure.

Executive Summary

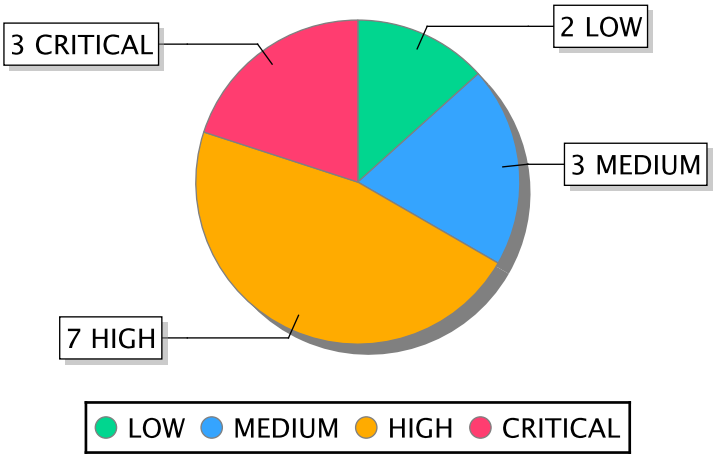
Assessment Overview and Recommendations

OWASP contracted Cel to perform a Penetration Test to identify security weaknesses, determine the impact to OWASP, document all findings in a clear and repeatable manner, and provide remediation recommendations.

Number of Findings per Category



Severity Overview of Findings



Comment: 0dd84537-6be7-468f-a4ad-6cf30d8fb7dc

Title:

Webserver Type

Description:

When looking at "Server" property of the repsonse header we can see that the application is running on a Cowboy HTTP Server.

Comment: f4901f6b-6814-450c-8734-7ff1b3eed9b0

Title:

Deprecated MIME Types

Description:

When looking through the main.js file of the webserver we can search for "allowedMimeType" and get presented with the following:
- application/pdf
- application/xml
- text/xml
- application/zip
- application/x-zip-compressed
- multipart/x-zip

Especially the upload of xml files can result in a XXE Attack or in a RCE.

Finding: ac45159b-4108-4ec2-b6aa-d3bfc5d597d2

Title:

Enumuration of Webserver

LOW

Description:

Running nmap against the Webserver we can find the following information about the installed services.
Interesting ports on 54.78.134.111:
- 993/tcp is running imaps
- 995/tcp is running pop3s
- 3128/tcp is runnung squid-http
- 8080/tcp is running http-proxy

Impact:

Webserver

Reproduction Steps:

Step 1:
Resolve IP-Address of Webserver (https://juice-shop.herokuapp.com/) through nslookup.

Step 2:
Scan the address that got returned from the DNS via nmap (nmap -sC -sV 54.78.134.111).

Step 3:
See what service runs on which port.

Mitigation:

No mitigation to avoid, minimize or compensate the finding found or needed.

Affected URL's / API's:

https://juice-shop.herokuapp.com/

Finding: 972b0cee-13e5-4267-ab5c-5b00c9657578

Title:

Admin Useraccount

HIGH

Description:

When looking through the application it is possible to find the censored e-mail of an user with an juice-shop mail (***der@juice.sh.op) that can be found on the "About Us" page by the customer feedback section.

Upon further investigating the product reviews the complete admin e-mail (admin@juice-sh.op) can be found in the review for the Apple Juice.

This account can now be used by an attacker to try to bruteforce into the account since the username is now known.

Impact:

Webserver only.

Reproduction Steps:

Step 1:
Look at the hompage.

Step 2:
Click on the "Apple Juice (1000ml)" Item.

Step 3:
Open the reviews.

You can now directly see the e-mail of the admin user.

Mitigation:

Censor important usernames of accounts with high privileges like seen on the "About Us" page by the customer feedback section.

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/about

Comment: 5514f0d3-7c80-4138-bf3e-56b515560f00

Title:

OWASP Juice Shop Architecture

Description:

In the frontend the popular Angular framework is used to create a so-called Single Page Application.
JavaScript is also used in the backend as the exclusive programming language: An Express application hosted in a Node.js server delivers the client-side code to the browser. It also provides the necessary backend functionality to the client via a RESTful API.
As an underlying database a light-weight SQLite was chosen, because of its file-based nature. Sequelize and finale-rest are used as an abstraction layer from the database.
As an additional data store, a MarsDB is part of the OWASP Juice Shop.
The application also offers user registration via OAuth 2.0 so users can sign in with their Google accounts.

Finding: 354c62b1-8f7f-4a65-9f1b-c4f6388f5506

Title:

Broken Access Control

HIGH

Description:

Security flaws are caused by fragilely implemented access rights (or non-well-thought access constructs).
Access control is based on:
- Confidentiality of the requested element
- Role or permissions of the requesting user
Flaws in access control can lead to:
- Unauthorized users can obtain, manipulate or delete important and sensitive data
Changing the bid inside the session storage in the frontend or intercepting the GET request for the basket and changing the id parameter results in getting the basket of another user (as long as the new id is valid).

Impact:

This does not just affect the frontend but also destroys the integrity of the data from the backend since you can see the basket of other users.

Reproduction Steps:

Step 1:
Login as any user.

Step 2:
Go to the basket page.

Step 3:
Open the browser console and change the bid value or intecept and manipulate the GET request for the basket.

Mitigation:

Decide for a matching access control model:
- Discretionary access control (DAC)
- Role-based access control (RBAC)
- Mandatory access control (MAC)
- Attribute-based access control (ABAC)
- Rule-based access control (RuBAC)

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/basket, https://juice-shop.herokuapp.com/rest/basket/{id}

Finding: 6edcdc7b-196e-4c72-bc74-7a8ae2ab3818

Title:

Sensitive Data Exposure

HIGH

Description:

Proper configuration of the single elements that make up an application architecture is important in order to prevent mistakes that might compromise the security of the whole architecture.

The web server or application server configuration takes an important role in protecting the contents of the site and it must be carefully reviewed in order to spot common configuration mistakes.

Accessing the Logfiles of the server is a problem that was encounterd.

Impact:

Webserver

Reproduction Steps:

Step 1:
Search for different sub directories on the webserver with tools like DirBuster.

Step 2:
Go to the sub directory /support/logs

Step 3:
Download the access.log.2023-07-19 file.

Mitigation:

Block access for users who are not authenticated and / or authorized to see the logs or other sub directories like the ftp content.

Affected URL's / API's:

https://juice-shop.herokuapp.com/support/logs, https://juice-shop.herokuapp.com/ftp/

Finding: 97daef3d-46be-43de-9950-7451da2e99c9

Title:

MEDIUM

Permitting default, weak, or well-known passwords

Description:

There may be authentication weaknesses because of:
- Automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Brute force or other automated attacks.
- Permitting default, weak, or well-known passwords, such as ”123456".
- Weak or ineffective credential recovery and forgot-password processes.
- Using plain text, encrypted, or weakly hashed passwords data stores.
- Missing or ineffective multi-factor authentication.
- Exposing session identifier in the URL.
- Reusing session identifier after successful login.

Impact:

Userdata

Reproduction Steps:

Step 1:
Go to the login page and then try to register a new user.

Step 2:
Fill out the form and type in a basic password like "lorem".

Step 3:
Click on "Register"

Mitigation:

Change the Switch for "Show password advice" to enforce these policies on creation and just give them the users as an suggestions.

More generally you should:
- Implement password validation and secure password guidelines
- Implement countermeasures against brute-force attacks
- Use best practices for session management
- Check secure password policies from "Hive Systems"

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/register

Finding: 16bf3a81-982a-445d-8a84-d0b151bd1f71

Title:

HIGH

Resetting Jim's Password

Description:

The password change and reset function of an application is a self-service password change or reset mechanism for users. This self-service mechanism allows users to quickly change or reset their password without an administrator intervening.
When passwords are changed they are typically changed within the application.
When passwords are reset they are either rendered within the application or emailed to the user. This may indicate that the passwords are stored in plain text or in a decryptable format.

When looking at the security question when creating an account we can see the options a user has. This information can be used to attack the web application, for example, through a brute force when resetting a password.

Impact:

Useraccount: jim@juice-sh.op

Reproduction Steps:

Step 1:
We can get to the email for jim when looking at the reviews for the "Green Smoothie" in the Juice Shop homepage.

Step 2:
Go to the login page and click on "Forgot your password?"

Step 3:
Enter jims email (jim@juice-sh.op) and click inside the "Security Question" Field.
We can now see that he question jim choose was "Your eldest siblings middle name?"

Step 4:
Since names are a simple property to find out if the user answered the question honestly.
We can use a list of the most popular names for males and females and brute-force the forgot password process with tools like BurpSuite or OWASP ZAP.

Mitigation:

To stop an attacker from brute-forcing anything you should
- Limit Specific Request Attempts
- Monitor IP addresses
- Use Two-Factor Authentication (2FA)
- Use CAPTCHAs
- Use Web Application Firewalls (WAFs)

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/forgot-password, https://juice-shop.herokuapp.com/rest/user/reset-password

Finding: 5924c1c6-348b-403c-af41-d5e0fab05c1b

Title:

SQLITE Error

MEDIUM

Description:

Provoked an error that is neither gracefully nor consistently handled.

Impact:

Webapplication and Node.js Server.

Reproduction Steps:

Step 1:
Go to Login.

Step 2:
Insert ' in username field and any string in password field.

Step 3:
Send the request and observe the error message [object object displayed].
Inside the Network traffic the response body for the login POST request with the "incomplete" SQL Injection returns to much information.

We can see that the errorcode is from SQLITE and get the sql query returned that got executed:
SELECT * FROM Users WHERE email =" ' " AND password = "randomString"

Mitigation:

Sanitize and validate Input Fields.
Use proper error handeling in Backend. Rule of thumb: Escape all user input...

Affected URL's / API's:

No specific URL's or API's affected.

Finding: 14e52a97-b147-4cd4-a0d5-d349bd9ca201

Title:

Ephemeral Accountant

MEDIUM

Description:

We logged into the application with an (non-existing) accountant acc0unt4nt@juice-sh.op with accountant-level permissions without ever registering that user and we created the needed user “out of the air”.

Impact:

Webapplication and Database Server.

Reproduction Steps:

Step X:
Enter the following sql syntax in the login field email and enter any sting in the password field:
' UNION SELECT * FROM (SELECT 15 as 'id', '' as 'username',
'acc0unt4nt@juice-sh.op' as 'email', '12345' as 'password',
'accounting' as 'role', '123' as 'deluxeToken',
'1.2.3.4' as 'lastLoginIp' ,
'/assets/public/images/uploads/default.svg' as 'profileImage',
'' as 'totpSecret', 1 as 'isActive',
'1999-08-16 14:14:41.644 +00:00' as 'createdAt',
'1999-08-16 14:33:41.930 +00:00' as 'updatedAt',
null as 'deletedAt')--

Mitigation:

No mitigation to avoid, minimize or compensate the finding found or needed.

Affected URL's / API's:

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

Finding: 271a2e85-5804-4b78-a12a-b722049321b3

Title:

HIGH

Admin Account SQL Injection for Login

Description:

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner.
Inside Login Form using the ' or TRUE-- Syntax will enable the user to login as the Admin.

Impact:

The active User-Session with Admin priviledges can affect the whole application.

Reproduction Steps:

Step 1:
Go to login page.

Step 2:
Enter ' or TRUE-- in the username field and enter a random password.

Step 3:
Click “Login”.

You will now be authenticated as the Juice Shop Admin.

Mitigation:

Preventing SQL Injections can be easily accomplished by adding: - Prepared statements
- Stored procedures
- Whitelist Input Validation
- Escaping all input, that could be user-supplied - Webapp Firewall

Affected URL's / API's:

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

Finding: 810138c0-dee4-42b3-8580-1fd0a0d92fa6

Title:

CRITICAL

Exfiltrated the entire DB schema definition via SQL Injection

Description:

An attacker would try to exploit SQL Injection to find out as much as possible about your database schema.
This subsequently allows much more targeted, stealthy and devastating SQL Injections.

Impact:

Database Server

Reproduction Steps:

Step 1:
Search for any product in the Juice Shop.

Step 2:
Look at the network traffic and copy the search request (https://juice-shop.herokuapp.com/rest/products/search?q=)

Step 3:
Run the request through sqlmap liek:
sqlmap -u http://0.0.0.0:3000/rest/products/search\?q\= --dbs --level=3 --risk=3

Step 4:
Run the request through sqlmap with schema flag like:
sqlmap -u http://0.0.0.0:3000/rest/products/search\?q\= --schema

Step 5:
Enterering the following string in the search field results in getting the the emails and password hashes of all users:
test ')) UNION ALL SELECT NULL,email,password,NULL,NULL,NULL,NULL,NULL,NULL from users--

Mitigation:

Preventing SQL Injections can be easily accomplished by adding: - Prepared statements
- Stored procedures
- Whitelist Input Validation
- Escaping all input, that could be user-supplied - Webapp Firewall

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/search?q=, https://juice-shop.herokuapp.com/rest/products/search?q=

Finding: 4e28eb62-2a59-471d-b1f8-2b3de54f541b

Title:

NoSQL Manipulation (Injection)

LOW

Description:

NoSQL Injection is different than classic SQL Injection, so I decided to broader my knowledge, reading A NoSQL Injection Primer (with Mongo) – Null Sweep article.

There is trick described, when author is bypassing logging page with simple $ne (not-equals) verb like.

Impact:

Mongo Database Server

Reproduction Steps:

Step 1:
Open any product in the Juice Shop homepage after logging in and write a review.

Step 2:
Look at the PUT request and change the request body from:
{"message":"test","author":"admin@juice-sh.op"}
to this:
{"id": { "$ne": -1 }, "message":"test"}

Step 3:
Send the request with the $ne (not-equals) verb.

Mitigation:

No mitigation to avoid, minimize or compensate the finding found or needed.

Affected URL's / API's:

https://juice-shop.herokuapp.com/rest/products/1/reviews

Finding: fa34ae0c-b230-4448-818b-e55e14d2ce38

Title:

NoSQL DoS Injection

CRITICAL

Description:

A denial-of-service (DoS) attack occurs when legitimate users are unable to access information systems, devices, or other network resources due to the actions of a malicious cyber threat actor. Services affected may include email, websites, online accounts (e.g., banking), or other services that rely on the affected computer or network. A denial-of-service condition is accomplished by flooding the targeted host or network with traffic until the target cannot respond or simply crashes, preventing access for legitimate users. DoS attacks can cost an organization both time and money while their resources and services are inaccessible.

NoSQL databases provide looser consistency restrictions than traditional SQL databases.
So basically we will try to invoke sleep(milliseconds) MongoDB method.

Impact:

Database Server

Reproduction Steps:

Step 1:
Open any product on the Juice Shop Homepage.

Step 2:
Take the GET request and change the product id parameter with sleep(1000).

Step 3:
See the Serverresponse be delayed by the sleep command because the server is "napping".

Mitigation:

NoSQL databases provide looser consistency restrictions than traditional SQL databases. By requiring fewer relational constraints and consistency checks, NoSQL databases often offer performance and scaling benefits. Yet these databases are still potentially vulnerable to injection attacks, even if they aren’t using the traditional SQL syntax.

Affected URL's / API's:

https://juice-shop.herokuapp.com/rest/products/sleep(1000)/reviews

Finding: b215d04c-fec9-4f75-8d83-89ba0c6d3e74

Title:

Deprectated B2B Interface File Upload Error

HIGH

Description:

Inside the complaint screen the user is able to upload a file that should only be ment to be a pdf.
Upon expection of the allowed MIME Types included in the main.js file we can see the following MIME Types being accepted by the application:
["application/pdf", "application/xml", "text/xml", "application/zip", "application/x-zip-compressed", "multipart/x-zip"]
Uploading a XML File results in the following error message that doesn't get handled gracefully by the frontend:
"Error: B2B customer complaints via file upload have been deprecated for security reasons (filename.xml)"

Impact:

This deprecated interface affects the frontend, backend and potentially the database depending on how the uploaded file is being handeled in the backend.

Reproduction Steps:

Step 1:
Login to the application with any user.

Step 2:
Go to complaint screen.

Step 3:
Write a small message in text field and upload any xml file before clicking on "Submit".
You will now get the error mentioned in the description.

Mitigation:

Adjust the allowed MIME Type in the frontend.

Other generic prevention methods include:
- Check your HTTP response headers
- Check your TLS configuration

Never configure wildcards in:
- CORS allowed origin header

- Redirect URI for OAuth/OIDC

Use Configuration Management:
- Hardening, Remove old configurations
- Proper Error Codes

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/complain

Finding: b2d779c5-150e-4dee-a40e-0f45b6027ea3

Title:

Weird Crypto

CRITICAL

Description:

Initially confined to the realms of academia and the military, cryptography has become ubiquitous thanks to the Internet. Common every day uses of cryptography include mobile phones, passwords, SSL, smart cards, and DVDs.

The proper and accurate implementation of cryptography is extremely critical to its efficiency. A small mistake in configuration or coding will result in removing a large degree of the protection it affords and rendering the crypto implementation useless against serious attacks.

Impact:

Tokens / Cookies

Reproduction Steps:

Step 1:
Login to the application with any valid user.

Step 2:
Look at the network traffic and copy the token that is part of the cookie header.

Step 3:
Decode the Base64 Token on an application like Cyberchef and copy the included passsword hash.

Step 4:
Analyse the hash to find aout the type through:
https://www.tunnelsup.com/hash-analyzer/

Step 5:
Unsalted MD4 and MD5 hashes are barely speedbumps to cracking passwords at this point in time, and should never be used.

Step 6:
With tools like Hashcat or John the Ripper we can now easily crack the password and get it in cleartext.

Mitigation:

Use more secure algorithms to encrypt sensible data.
For information like passwords it is recommended to use salting and strong and slow hashing algorithms like:
- Argon2
- Bcrypt

Affected URL's / API's:

No specific URL's or API's affected.

Finding: 19521078-aef5-4505-8b1f-958e75bd3fd1

Title:

HIGH

Searchbar DOM XSS

Description:

DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way.
Adding <iframe src="javascript:alert(`xss`)"> in the search bar of the header results here in the XSS Vulnerability.

Impact:

Generally there are three kinds of XXS:
1. DOM-Based Cross-Site Scripting
2. Reflected Cross-Site Scripting
3. Persistent Cross-Site Scripting

The found XSS only impacts the Webapplication itself.

Reproduction Steps:

Step 1:
Click on the search field of the header.

Step 2:
Enter <iframe src="javascript:alert(`xss`)">

Step 3:
Press ENTER to exucute the query.

You will now get a PopUp because the javascript code was executed in the browser.

Mitigation:

- Do NOT put untrusted data into templates & SSR
- Use strict input validation & strong typing (server-side)
- Contextual Output Encoding
- Sanitizing Input Fields
- Content Security Policies
- Trusted Types
- Protect Session Cookie (HTTPOnly)

Affected URL's / API's:

https://juice-shop.herokuapp.com/#/search?q=%3Ciframe%20src%3D%22javascript:alert('xss')%22%3E

Appendencies

Findings Severities

Each finding has been assigned a severity rating of critical high, medium, or low. The rating is based off of an assessment of the priority with which each finding should be viewed and the potential impact each has on the confidentiality, integrity, and availability.

Risk Matrix

The risk matrix is used to assess the potential damage of a hazard, based on the likelihood and severity factors. The likelihood and severity scores are multiplied to obtain a score value. This score is looked up in the risk ranges to determine the risk level. An example of a hazard risk matrix is given below:

Risk score	Risk level category	*Likelihood*				
1 to 4	Low					
5 to 10	Medium					
11 to 18	High	Rare (1)	Unlikely (2)	Possible (3)	Likely (4)	Almost certain (5)
19 to 25	Critical					
Severity	Catastrophic (5)	Medium	Medium	High	Critical	Critical
	Major (4)	Low	Medium	High	High	Critical
	Moderate (3)	Low	Medium	Medium	High	High
	Minor (2)	Low	Low	Medium	Medium	Medium
	Insignificant (1)	Low	Low	Low	Low	Medium

Example, if Likelihood = Possible (3) and Severity = Major (4), the risk level is determined by severity * likelihood, which is 3*4 = 12. The score 12 falls in 'High' risk range.

Rating	Severity Rating Definitions
Critical	Exploitation of the technical or procedural vulnerability will cause substantial harm. Significant political, financial, and/or legal damage is likely to result. The threat exposure is critical, and a publicly available mechanism exists to exploit the vulnerability. Security controls are not effectively implemented to reduce the severity of impact if the vulnerability were exploited.
High	Exploitation of the technical or procedural vulnerability will cause substantial harm. Significant political, financial, and/or legal damage is likely to result. The threat exposure is high, thereby increasing the likelihood of occurrence. Security controls are not effectively implemented to reduce the severity of impact if the vulnerability were exploited.
Medium	Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity, and/or availability of the system, application, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment. The threat exposure is moderate-to-high, thereby increasing the likelihood of occurrence. Security controls are in place to contain the severity of impact if the vulnerability were exploited, such that further political, financial, or legal damage will not occur. - OR - The vulnerability is such that it would otherwise be considered High Risk, but the threat exposure is so limited that the likelihood of occurrence is minimal.
Low	Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The Confidentiality, Integrity and Availability (CIA) of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment. The threat exposure is moderate-to-low. Security controls are in place to contain the severity of impact if the vulnerability were exploited, such that further political, financial, or legal damage will not occur. - OR - The vulnerability is such that it would otherwise be considered Medium Risk, but the threat exposure is so limited that the likelihood of occurrence is minimal.