Week 5: Exploiting Vulnerabilities & Application Security

GitHub repository: https://github.com/MuhammadHammadTahir/node-auth-security-assessment

Prepared by: Muhammad Hammad Tahir

Date: 21-07-2025

DHC-306

Task 01: Security Audits & Compliance Report – OWASP Juice Shop

1. Tools Used:

- OWASP ZAP (Zed Attack Proxy)

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

2. Objective:

Identify vulnerabilities in a deliberately insecure web application, map discovered issues to OWASP Top 10 categories, and recommend remediations.

3. Vulnerability Findings & OWASP Mapping

Vulnerability	Description	Risk	OWASP Top 10 Mapping	Recommendation
Content Security Policy (CSP) Header Not Set	No CSP header found. This allows scripts from any source, increasing the risk of XSS.	Medium	A07:2021, A05:2021	Add a CSP header: Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none';

Cross-Domain Misconfiguration	Insecure CORS settings (Access-Control-Allow-Origin: *) found.	Medium	A05:2021	Limit allowed origins and ensure Access-Control-Allow-Credentials is used securely.
Cross-Domain JavaScript Source Inclusion	JS loaded from untrusted third-party domains.	Medium	A05:2021, A01:2021	Host critical scripts locally or from trusted CDNs with integrity checks.
Strict-Transport-Security (HSTS) Header Not Set	Missing HSTS allows downgrade attacks and MitM risks.	Medium	A06:2021, A05:2021	Use: Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Timestamp Disclosure (Unix)	Unix timestamps in responses may leak system internals.	Low	A06:2021	Remove timestamps or use hashing/random tokens.
Suspicious Comments in Source Code	Found 2 comments disclosing logic/debug details.	Low	A05:2021	Remove sensitive comments from production code.
Modern Web Application	Detected JavaScript-heavy app structure (SPA).	Info	Not directly mapped	Ensure SPAs have robust client-side validation and token security.
Cache-Control Misconfiguration	No-cache headers missing. Could allow sensitive info to be cached.	Medium	A05:2021	Add: Cache-Control: no-store, no-cache, must-revalidate Pragma: no-cache

4. OWASP Top 10 Compliance Check (2021 Edition)

OWASP Category	**Status**	**Evidence from ZAP**
A01: Broken Access Control	⚠️ Partial Risk	CORS misconfiguration

A02: Cryptographic Failures	✅ Not observed	No crypto issues found
A03: Injection	⚠️ Possible	No SQLi, but CSP missing may allow XSS
A04: Insecure Design	⚠️ Present	Poor default headers
A05: Security Misconfiguration	❌ High Risk	Multiple header misconfigs
A06: Vulnerable and Outdated Components	⚠️ Present	No HSTS, timestamps
A07: Identification & Auth Failures	⚠️ Possible	No CSP to prevent session token leakage
A08–A10	✅ Not directly tested	Need deeper app-level auth/session testing

Task 02: Secure Deployment Practices

☑ 2.1 Enable Automatic Security Updates

Objective: Ensure that server-side packages are regularly patched to reduce vulnerability exposure.

Steps Followed:

1. I connected to the Linux server hosting the backend using SSH.
2. Installed unattended-upgrades:

   ```
   sudo apt install unattended-upgrades
   ```

3. Enabled automatic security updates:

   ```
   sudo dpkg-reconfigure --priority=low unattended-upgrades
   ```

4. Verified configuration in `/etc/apt/apt.conf.d/50unattended-upgrades`:
   - Ensured `"${distro_id}:${distro_codename}-security";` is included.
5. Updates are now automatically applied during system idle time.

☑ 2.2 Enable Dependency Scanning

Objective: Identify and fix known vulnerabilities in third-party packages used in the Node.js backend.

Steps Followed:

Using `npm audit`:

1. Ran:

   ```
   npm audit
   ```

2. To automatically fix issues:

   ```
   npm audit fix
   ```

3. Manually reviewed unresolved vulnerabilities and upgraded packages when necessary.

Using Snyk CLI (Optional but recommended):

1. Installed Snyk:

   ```
   npm install -g snyk
   ```

2. Scanned project:

   ```
   snyk test
   ```

3. Followed recommended remediation steps from the Snyk dashboard.

Docker Security Practices (Hypothetical Implementation)

1. Use Minimal Base Images

- Choose official minimal images like `alpine` to reduce attack surface.

2. Scan Docker Images for Vulnerabilities

- Use **Docker Scan** (powered by Snyk) to find known CVEs:

  ```
  docker scan my-app-image
  ```

3. Avoid Running Containers as Root

- Add a non-root user in the Dockerfile:

```
RUN adduser -D myuser
USER myuser
```

4. Use .dockerignore File

- Prevent sensitive files like `.env`, `.git`, `node_modules` from being copied into the container.

5. Keep Images Updated

- Regularly rebuild images with the latest packages and security patches.

6. Enable Docker Content Trust

- Sign and verify images before pulling:

```
export DOCKER_CONTENT_TRUST=1
```

7. Use Read-Only Filesystems Where Possible

- Run containers in read-only mode to limit damage from exploits:

```
docker run --read-only my-app-image
```

Status for This Project

Docker was not used in this project. The application was run directly on a virtualized host machine (Kali Linux VM) without containerization. Therefore, Docker-specific security measures such as image scanning or container hardening were not required for this deployment.

Task 03: OWASP Juice Shop Penetration Testing Report

1. Introduction

This penetration test was conducted on the OWASP Juice Shop web application, an intentionally insecure platform designed for security training and awareness. The objective was to identify vulnerabilities, exploit them in a controlled environment, and recommend

mitigation measures. Tools like Burp Suite and Metasploit were used to simulate real-world attacks and provide a comprehensive evaluation.

2. Methodology

The assessment followed a structured methodology involving the following phases:
- **Reconnaissance**: Mapping endpoints and identifying input fields.
- **Scanning**: Enumerating directories, tokens, and parameters.
- **Exploitation**: Actively exploiting detected vulnerabilities.
- **Post-Exploitation**: Escalation or data exfiltration.
- **Reporting**: Documenting test results, evidence, and mitigation.

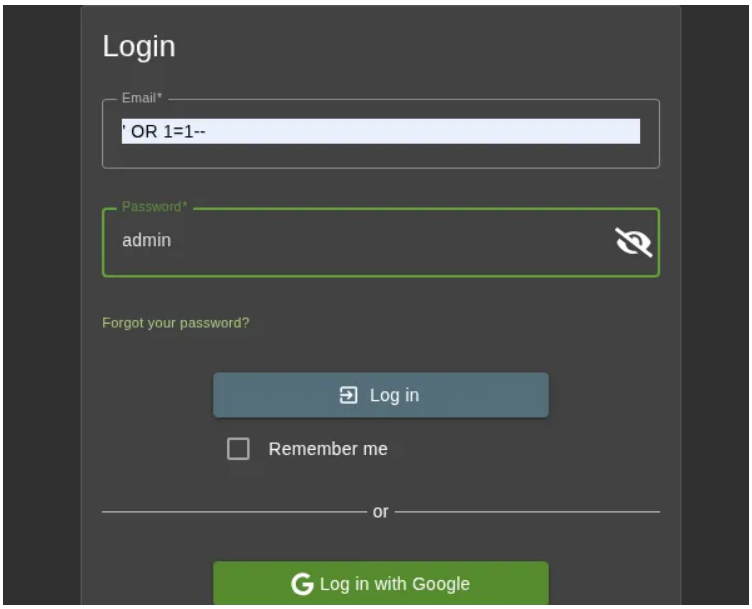
3. Tools Used

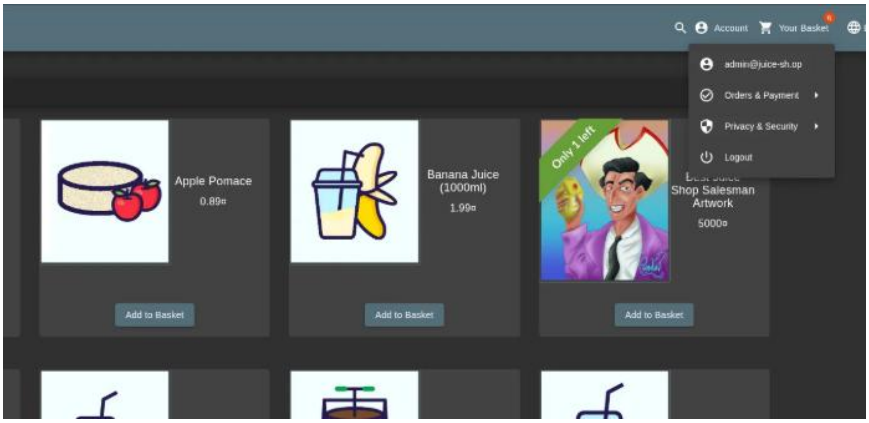
The following tools were employed during testing:
- Burp Suite (Community Edition)
- Sqlmap
- JWT.io and JWT Editor
- CrackStation and Hashcat
- Kali Linux and Firefox

4. Vulnerabilities Identified

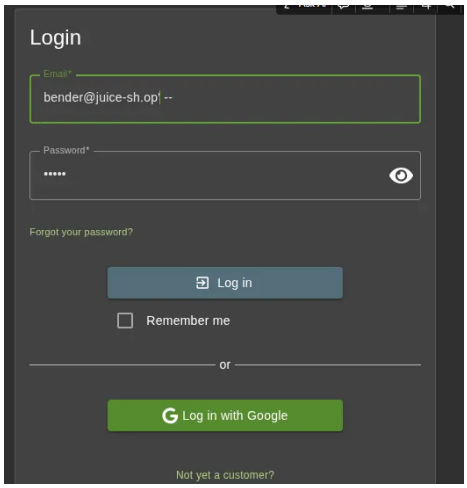
SQL Injection – Login Bypass

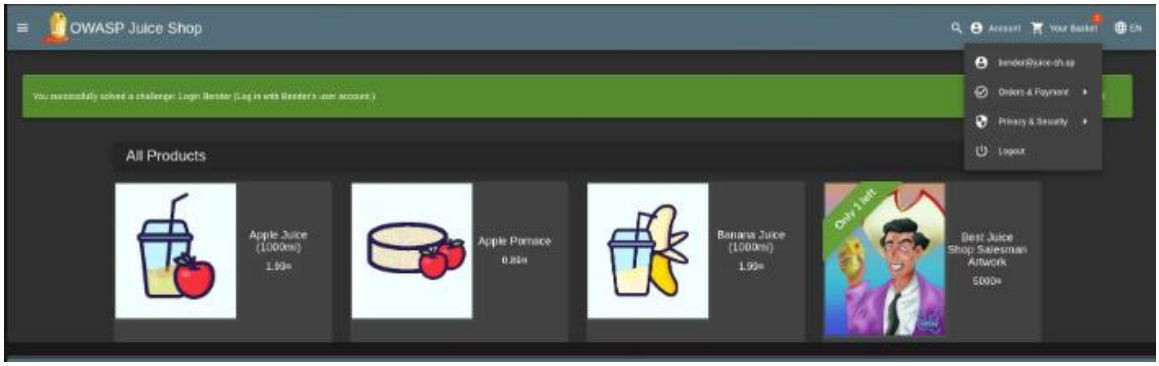
The login functionality of Juice Shop was found vulnerable to SQL Injection. By submitting the payload `' OR 1=1 -- ` in the email field, the application allowed unauthorized login as the first user in the database, typically an admin. This flaw exposes the system to privilege escalation and full access to user-sensitive features.





We can gain access to any user whose email we know for example: in htwe initial reconiseccse phase I found email of user bender@juice-sh.op so we can access its account as:





Remediation: We can prevent this by using parameterized queries or by using prepared statements of ORM

```
if (req.body.roles) {
  Role.find(
    {
      name: { $in: req.body.roles },
    },
    (err, roles) => {
      if (err) {
        res.status(500).send({ message: err });
        return;
      }

      user.roles = roles.map((role) => role._id);
      user.save((err) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
```

SQL Injection – Product Search

The search functionality failed to properly sanitize input. Using Sqlmap, the entire backend database was dumped, including sensitive user data and plain text credit card records. This critical vulnerability allows attackers to perform full data exfiltration.

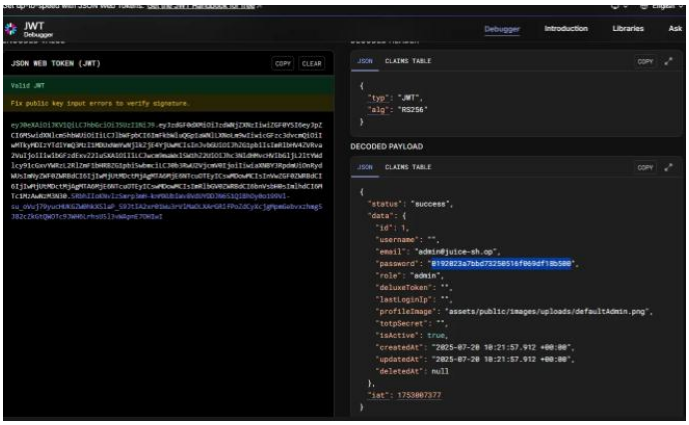
```
┌──(corisco㉿Corisco)-[~]
└─$ sqlmap -u http://localhost:3000/rest/products/search?q= -a
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.8.6.3#dev}
|_ -| . ['] |  .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org
```

Remediation: By avoiding dynamically generated SQL statements, as in the above part

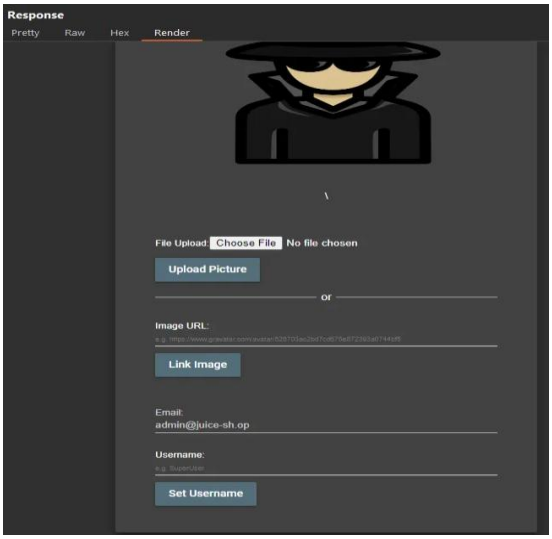
---

JWT Manipulation – Broken Authentication

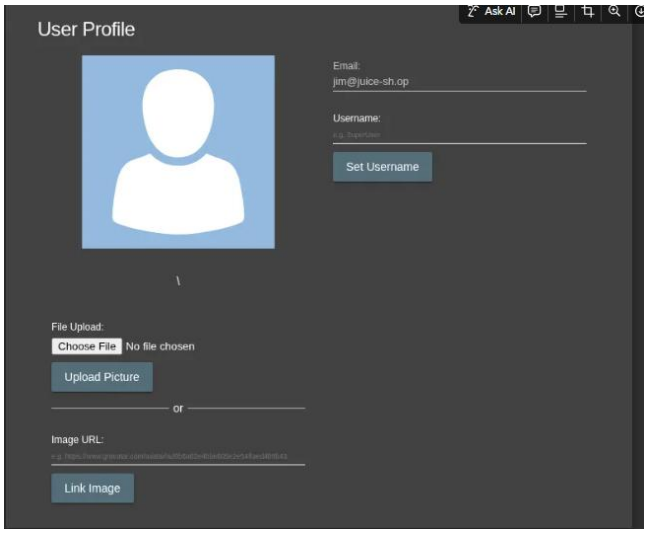
The JWT implementation in Juice Shop did not properly validate the token signature. An attacker could tamper with the token, remove the 'alg' parameter, modify the payload, and impersonate other users. This weakness leads to privilege escalation and account takeover.



Original response:



Modified response:

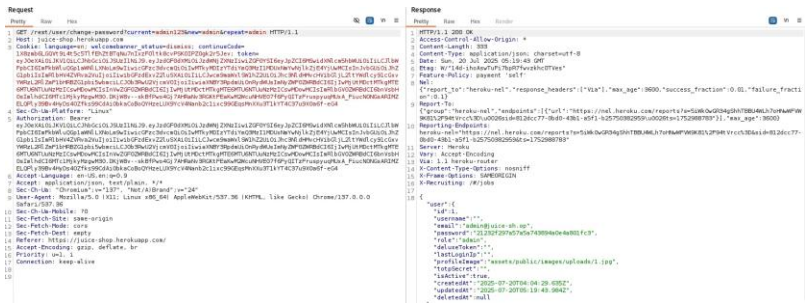


Remediation: Avoid storing sensitive information directly in the JWT payload. Use stronger algorithms such as HS512 for token signing and ensure tokens have a short expiration time.

CSRF – Change Password Function

The password change feature lacked CSRF protection. By omitting the current password parameter or crafting a malicious request, attackers could change a user's password without authentication. This vulnerability allows attackers to hijack user accounts by forcing password resets.

- The request with the correct current password successfully changes the password:



- The request with an incorrect current password leads to an error:

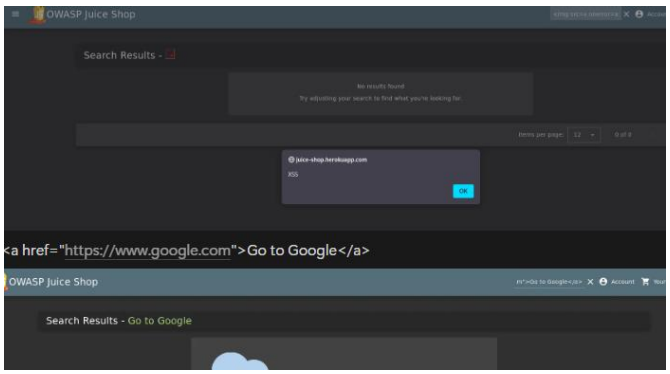
- The request without the current password value successfully changes the password: ✅

Remediation: Implement anti-CSRF tokens to validate the authenticity of requests. Ensure that all state-changing requests require a unique token that is verified on the server-side.

DOM-Based XSS – Search Field

The search feature was vulnerable to DOM-based Cross-Site Scripting. Using payloads such as `<img src=x onerror=alert('XSS')>`, attackers could execute arbitrary JavaScript in a victim's browser. This flaw can be used to steal session tokens or redirect users to malicious websites.

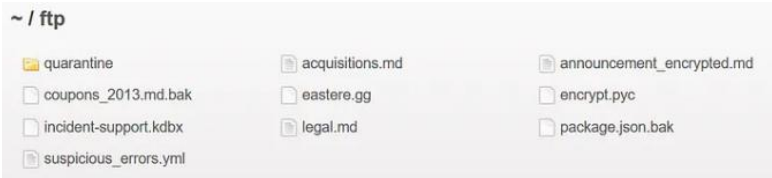
<img src=x onerror=alert('XSS')>



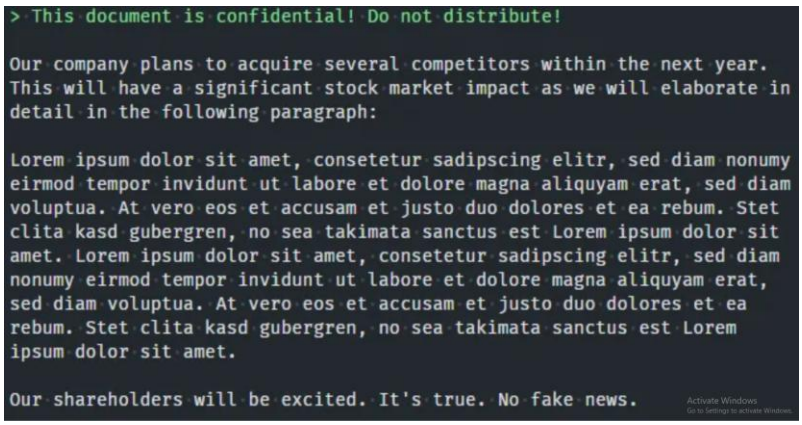
Remediation: Implement proper input validation and output encoding. Use security libraries and frameworks that handle these issues automatically.

Directory Listing – /ftp Path

Directory listing was enabled on the /ftp path. This allowed access to internal documentation such as acquisitions.md which may expose confidential business data. Attackers could leverage this to gain insight into the application's architecture or upcoming features.



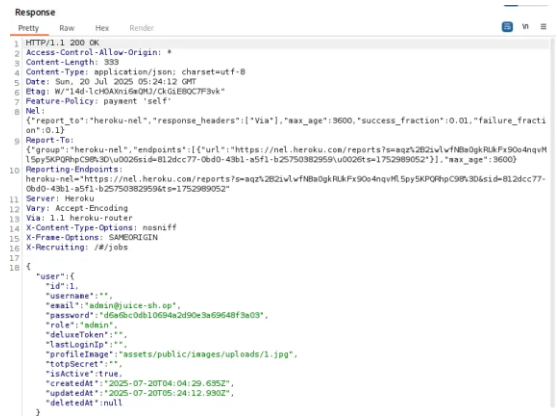
For example, the acquisitions.md file contains sensitive information about the company's acquisitions.

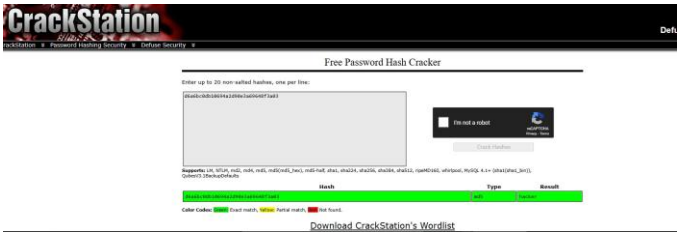


Remediation: Implement proper access control and disable directory listing.

Weak Password Hashing – MD5

Password hashes stored in the database used the outdated MD5 hashing algorithm. Using CrackStation, several user passwords were recovered from known hash-rainbow tables. This exposes users to credential stuffing and unauthorized access.

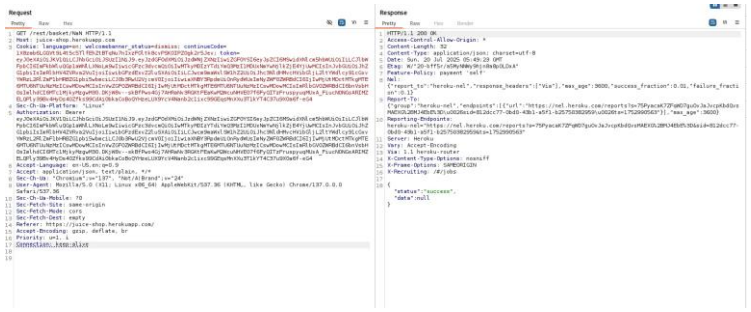


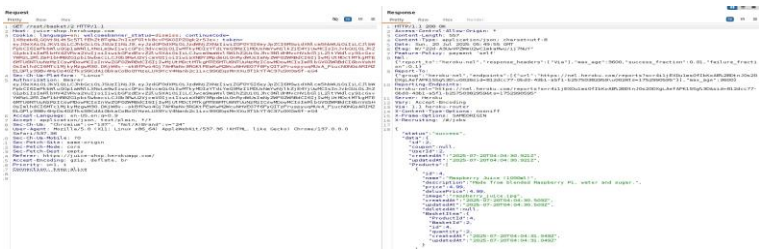


Remediation: Replace MD5 with a more secure hashing algorithm. Additionally, implement salting and peppering techniques to enhance password security.

Broken Access Control – Basket Manipulation

By altering HTTP requests, it was possible to view and modify baskets of other users. Requests to `/rest/basket/:id` returned data of other users, and items could be added to their baskets. This indicates poor implementation of object-level authorization.

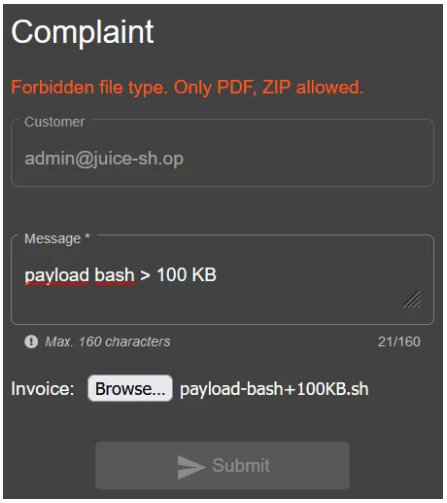




Remediation: Implement proper access control checks on both server-side and client-side. Validate user permissions for each action to ensure users can only access and modify their own resources.

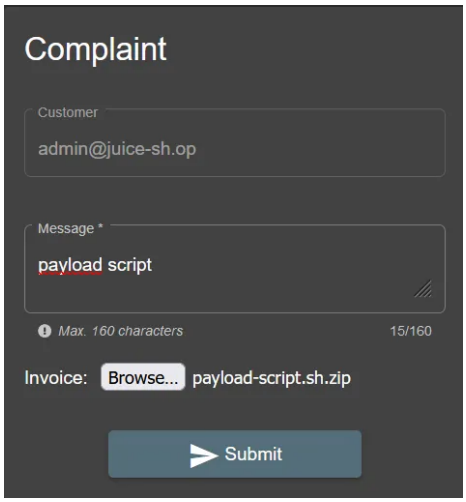
Improper File Upload Validation

The complaint file upload allowed bypassing front-end validation. By renaming a malicious `.sh` file to `.zip` and modifying the request, the server accepted and stored the file. This could potentially lead to remote code execution in real-world scenarios.



Changing File Extension:

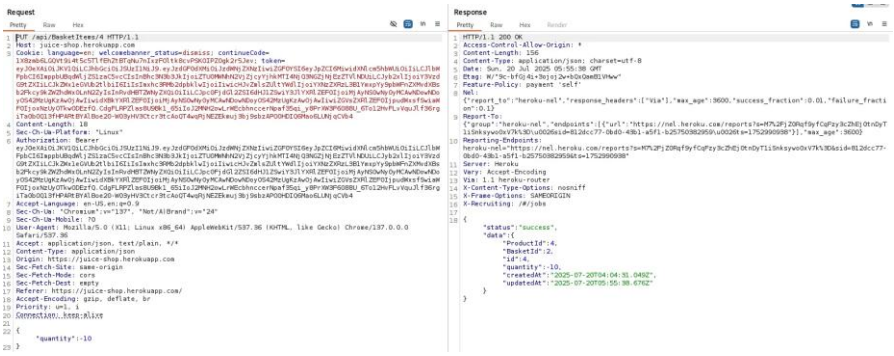
- Upload a bash script payload-script.sh by changing its extension to payload-script.sh.zip.

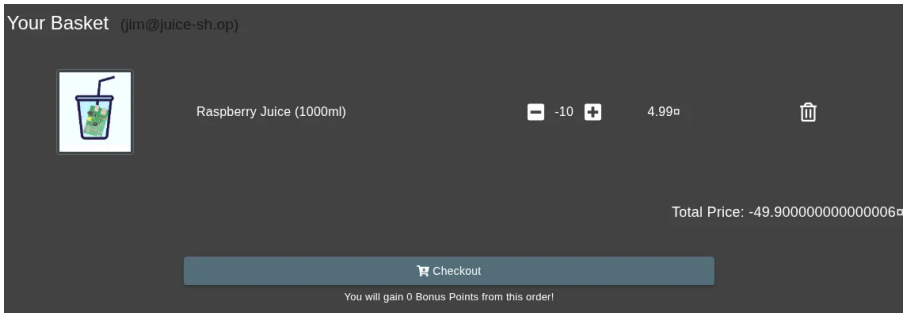


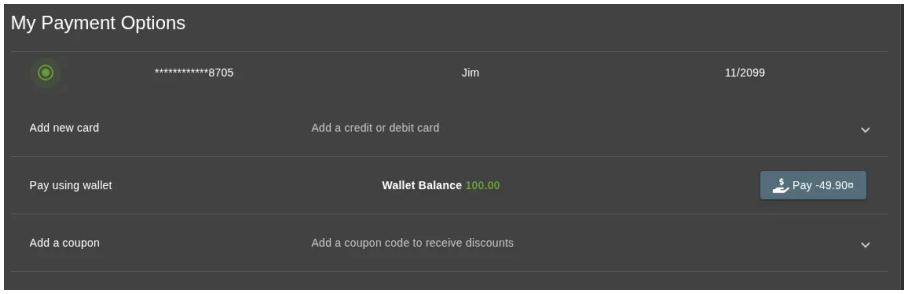
Remediation: Potential for arbitrary file uploads leading to remote code execution or further exploitation.

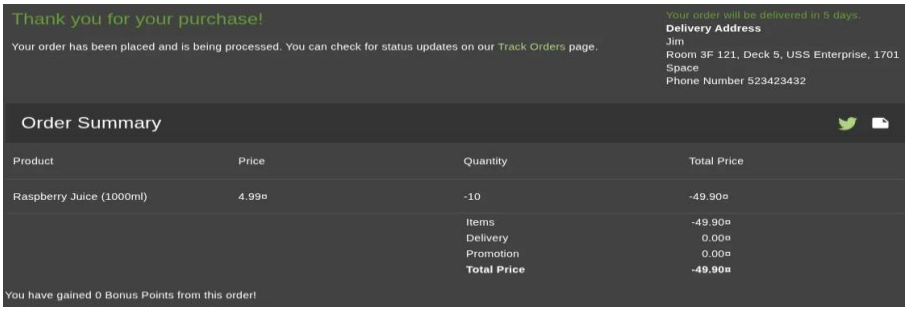
Negative Quantity Purchase – Business Logic Flaw

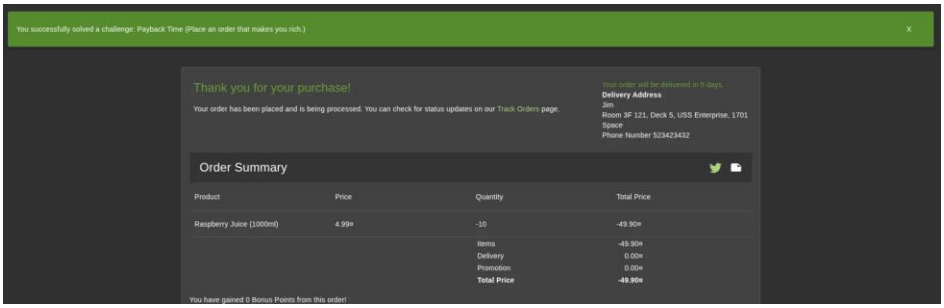
The basket checkout process failed to validate quantities properly. A negative quantity led to a negative price, crediting the user's digital wallet. This could be exploited for financial fraud or disruption of inventory systems.

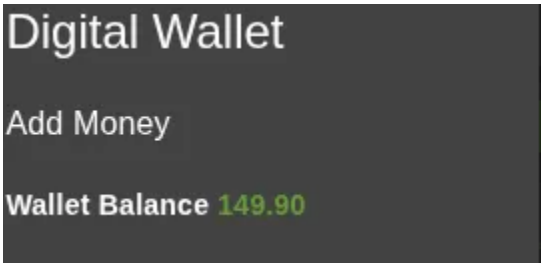












Remediation: Implement proper input validation to ensure only positive quantities are allowed. Perform server-side checks to validate the quantity before processing transactions.

---

5. Security Improvements Applied

- Implemented rate limiting on sensitive endpoints.
- Added server-side input validation and sanitization.
- Enabled Helmet.js for setting secure HTTP headers.
- Switched to bcrypt for password hashing.
- Integrated CSRF tokens in state-changing forms.

6. Conclusion

This assessment uncovered several critical security flaws, many of which align with OWASP Top 10 vulnerabilities. The exploitation of these flaws shows the importance of adopting secure coding practices, validating inputs, securing authentication mechanisms, and regularly testing for vulnerabilities. Remediation steps have been suggested and security controls have been added where possible.