DASHSEC

Penetration Test Report

Target: OWASP Juice Shop v19.1.0 | http://192.168.86.249:3000
Date: 2026-02-18
Overall Risk Level: CRITICAL
---

1. Executive Summary

A penetration test was conducted against the OWASP Juice Shop web application running on `192.168.86.249:3000`. The assessment identified and successfully exploited **13 vulnerabilities**, including multiple critical flaws that enabled complete application compromise.

The most severe finding was a **SQL injection vulnerability in the login endpoint** that granted immediate, unauthenticated administrative access. This was compounded by **publicly exposed MD5 password hashes** for all users (including administrators), a **mass assignment flaw** allowing any user to register with admin privileges, and **broken access controls** enabling unauthorized access to other users' data.

Through chaining these vulnerabilities, full compromise was achieved: administrative access was obtained, all 31 user accounts were enumerated, the admin password was cracked, confidential business documents were downloaded, and a persistent backdoor admin account was created all without any prior credentials.

Top Priority Remediations

1. **Immediately patch the SQL injection** in the login endpoint (`/rest/user/login`) -- this alone enables full admin takeover.
2. **Remove password hashes** from the public `/rest/memories` API response -- this leaks credentials for every user.
3. **Enforce server-side role validation** on user registration (`/api/Users`) -- currently allows anyone to create admin accounts.
4. **Implement proper authorization checks** across all API endpoints to prevent IDOR and unauthorized data access.
5. **Add authentication** to administrative and infrastructure endpoints (`/rest/admin/*`, `/metrics`, `/ftp/`).

---

2. Confirmed Vulnerabilities

2.1 SQL Injection Authentication Bypass

Attribute	Detail
Severity	CRITICAL
Type	SQL Injection
Location	`POST /rest/user/login`

Description:
The login endpoint directly concatenates user-supplied input into a SQL query without parameterization. By injecting a SQL payload into the `email` field, the authentication logic is bypassed entirely, returning a valid admin session token.

Business Impact:
Any unauthenticated attacker can log in as the site administrator without knowing any credentials. This grants full control over the application, including access to all user data, orders, and administrative functions.

Evidence:
Request payload:

```
{"email": "admin@juice-sh.op' OR 1=1--", "password": "any"}
```

Response (truncated):

```
{
  "authentication": {
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9...",
    "umail": "admin@juice-sh.op",
    "bid": 1
  }
}
```

The returned JWT token grants full admin-level access to all protected endpoints.

Remediation:
Replace all raw SQL query construction with parameterized queries (prepared statements). For the SQLite backend:

```
// VULNERABLE
const query = `SELECT * FROM Users WHERE email='${req.body.email}'`;

// FIXED
const query = `SELECT * FROM Users WHERE email = ?`;
db.get(query, [req.body.email], callback);
```

Additionally, implement a Web Application Firewall (WAF) rule to detect common SQL injection patterns as a defense-in-depth measure.

2.2 SQL Injection — Product Search Data Extraction

Attribute	Detail
Severity	HIGH
Type	SQL Injection
Location	`GET /rest/products/search?q=`

Description:
The product search parameter is concatenated directly into a SQL query. Injecting SQL syntax bypasses search filtering and can be leveraged for UNION-based data extraction.

Business Impact:
Attackers can extract arbitrary data from the database, including user credentials, order history, and any other stored information. Error messages also disclose the database engine (SQLite) and internal file paths.

Evidence:

- Normal search (`?q=apple`): Returns 1 product.
- Injected search (`?q=apple')) OR 1=1--`): Returns all 47 products.
- Error disclosure: `SQLITE_ERROR: near "')" syntax error` confirms injectable query and database type.

Remediation:
Use parameterized queries for the search function. Suppress verbose database error messages in production by implementing generic error responses.

2.3 Mass Assignment Admin Privilege Escalation

Attribute	Detail
Severity	CRITICAL
Type	Broken Access Control / Mass Assignment
Location	`POST /api/Users`

Description:
The user registration endpoint binds all incoming JSON properties directly to the user model without filtering. An attacker can include the `role` field in the registration request to assign themselves administrative privileges.

Business Impact:
Anyone can create an administrator account at will, completely bypassing the authorization model. This provides a persistent backdoor that survives password resets or patches to the SQL injection vulnerability.

Evidence:
Request:

```
{
  "email": "hacker@example.com",
  "password": "Hacker123!",
  "passwordRepeat": "Hacker123!",
  "securityQuestion": {"id": 1},
  "securityAnswer": "test",
  "role": "admin"
}
```

Response:

```
{
  "status": "success",
  "data": {
    "id": 33,
    "email": "hacker@example.com",
    "role": "admin",
    "isActive": true
  }
}
```

Remediation:
Implement an allowlist of permitted fields during user registration. Never allow `role`, `isActive`, or other privileged fields to be set by client input:

```
// Allowlist approach
const allowedFields = ['email', 'password', 'securityQuestion', 'securityAnswer'];
const userData = pick(req.body, allowedFields);
userData.role = 'customer'; // Server-enforced default
```

2.4 Sensitive Data Exposure — Public Password Hash Leakage

Attribute	Detail
Severity	CRITICAL
Type	Sensitive Data Exposure
Location	`GET /rest/memories` (unauthenticated)

Description:
The memories API endpoint returns associated user objects that include MD5 password hashes, email addresses, and role information -- all without requiring authentication.

Business Impact:
Any visitor can harvest every user's password hash. Because MD5 is cryptographically weak, these hashes are trivially crackable. The admin password (`admin123`) was cracked in seconds, confirming that real credential compromise is achievable.

Evidence:
Unauthenticated response (truncated):

```
{
  "data": [
    {
      "User": {
        "email": "admin@juice-sh.op",
        "password": "0192023a7bbd73250516f069df18b500",
        "role": "admin"
      }
    },
    {
      "User": {
        "email": "bjoern.kimminich@gmail.com",
        "password": "6edd9d726cbdc873c539e41ae8757b8c",
        "role": "admin"
      }
    }
  ]
}
```


The hash `0192023a7bbd73250516f069df18b500` was confirmed as MD5 of `admin123`, and login was verified with those credentials.

Remediation:

1. **Immediately** remove password, role, and other sensitive fields from the `/rest/memories` API response. Use a DTO/serializer that only exposes necessary fields (e.g., `id`, `caption`, `imagePath`).
2. **Migrate from MD5** to a modern, salted hashing algorithm such as bcrypt or Argon2id.
3. **Force a password reset** for all users, as existing hashes must be considered compromised.

2.5 Stored Cross-Site Scripting (XSS) Product Reviews

Attribute	Detail
Severity	HIGH
Type	Stored XSS
Location	`GET /rest/products/{id}/reviews` (and the corresponding review submission endpoint)

Description:
Product reviews accept and render arbitrary HTML/JavaScript without sanitization or encoding. Malicious scripts persist in the database and execute in the browser of any user who views the affected product page.

Business Impact:
An attacker can inject scripts that steal session tokens (JWTs stored in localStorage), redirect users to phishing pages, or perform actions on behalf of authenticated users -- including administrators viewing reviews.

Evidence:
Existing stored payload found in product 1 reviews:

```
{
  "message": "<script>alert(\"XSS\")</script>",
  "author": "xss@test.com"
}
```

Remediation:

1. Implement output encoding (HTML entity encoding) for all user-supplied content rendered in the browser.
2. Implement a Content Security Policy (CSP) header that restricts inline script execution:

```
Content-Security-Policy: default-src 'self'; script-src 'self'
```

3. Sanitize input server-side using a library such as DOMPurify before storing reviews.

2.6 Insecure Direct Object Reference (IDOR) Shopping Baskets

Attribute	Detail
Severity	HIGH
Type	Broken Access Control / IDOR
Location	`GET /rest/basket/{id}`

Description:
The basket endpoint uses a sequential numeric ID to retrieve shopping basket contents. Any authenticated user can access any other user's basket by changing the ID parameter; the server does not validate that the requesting user owns the requested basket.

Business Impact:
Attackers can view and potentially modify any customer's shopping cart. This exposes purchase intent data, product preferences, and could enable order manipulation or fraud.

Evidence:
Using an admin token, accessing another user's basket (ID 2):

```
{
  "id": 2,
  "UserId": 2,
  "Products": ["Raspberry Juice (1000ml)"]
}
```

Remediation:
Enforce ownership validation on every basket request:

```
app.get('/rest/basket/:id', authMiddleware, (req, res) => {
  if (req.params.id !== req.user.bid) {
    return res.status(403).json({ error: 'Access denied' });
  }
  // proceed with basket retrieval
});
```

2.7 Information Disclosure — Public Application Configuration

Attribute	Detail
Severity	HIGH
Type	Information Disclosure / Security Misconfiguration
Location	`GET /rest/admin/application-configuration` (unauthenticated)

Description:
The application's full runtime configuration is accessible without any authentication. This includes server settings, challenge metadata, product pricing, security question-answer mappings, and internal domain information.

Business Impact:
Attackers gain a complete blueprint of the application's internal workings, dramatically reducing the effort needed to discover and exploit other vulnerabilities. Security question answers embedded in the configuration can be used for account takeover via password reset.

Evidence (truncated):

```
{
  "server": { "port": 3000, "baseUrl": "http://localhost:3000" },
  "application": { "domain": "juice-sh.op", "name": "OWASP Juice Shop" },
  "challenges": {
    "overwriteUrlForProductTamperingChallenge": "https://owasp.slack.com",
    "xssBonusPayload": "<iframe width=\"100%\" height=\"166\"...>"
  }
}
```

Remediation:
Restrict this endpoint to authenticated admin users only, or remove it from production entirely. If needed for internal tooling, place it behind a separate authenticated administrative interface.

2.8 Directory Listing & Unauthorized File Access

Attribute	Detail
Severity	HIGH
Type	Security Misconfiguration / Information Disclosure
Location	`/ftp/`, `/support/logs/`, `/encryptionkeys/`

Description:
Multiple directories have directory listing enabled and serve files directly without authentication. The `/ftp/` directory contains confidential business documents, backup configuration files, and a KeePass password database.

Business Impact:
Confidential acquisition plans, historical coupon data, and application dependency lists are publicly downloadable. The KeePass database could contain credentials for other systems if the master password is weak.

Evidence:
Contents of `/ftp/acquisitions.md`:

```
# Planned Acquisitions
> This document is confidential! Do not distribute!
Our company plans to acquire several competitors within the next year...
```

Remediation:

1. Disable directory listing on all directories.
2. Remove or relocate sensitive files from publicly accessible paths.
3. Implement authentication and authorization for any file-serving endpoint.
4. Add server configuration to block access:

```
location /ftp/ { deny all; }
location /support/logs/ { deny all; }
location /encryptionkeys/ { deny all; }
```

2.9 Null Byte Injection File Extension Bypass

Attribute	Detail
Severity	MEDIUM
Type	Improper Input Validation
Location	`/ftp/` file download handler

Description:
The file download handler validates file extensions (allowing only `.md` and `.pdf`) but is vulnerable to null byte injection. By appending `%2500.md` to a restricted filename, the validation is bypassed while the operating system truncates the filename at the null byte, serving the original restricted file.

Business Impact:
Attackers can download any file in the `/ftp/` directory regardless of extension restrictions, including backup files (`.bak`) that may contain sensitive configuration, credentials, or historical data.

Evidence:

- Direct access to `/ftp/package.json.bak` returns: `"Only .md and .pdf files are allowed!"`
- Accessing `/ftp/package.json.bak%2500.md` successfully returns the file, revealing outdated dependencies with known vulnerabilities (e.g., `express-jwt: 0.1.3`, `js-yaml: 3.10`).

Remediation:
Sanitize filenames by stripping null bytes and any non-printable characters before validation:

```
const filename = req.params.file.replace(/\0/g, '');
```

Additionally, use an allowlist of specific downloadable files rather than relying on extension-based filtering.

2.10 Business Logic Flaw Negative Quantity Acceptance

Attribute	Detail
Severity	MEDIUM
Type	Business Logic Flaw
Location	`POST /api/BasketItems`

Description:
The shopping basket accepts negative product quantities. While there is validation preventing quantities above the per-user limit (e.g., 5), there is no validation for negative values. This can result in negative line item totals, reducing the overall order price.

Business Impact:
An attacker could manipulate order totals to pay less than the actual price -- or potentially receive a credit/refund. This represents a direct financial loss vector.

Evidence:
Request:

```
{"ProductId": 9, "BasketId": 1, "quantity": -100}
```

Response:

```
{
  "status": "success",
  "data": { "id": 9, "ProductId": 9, "BasketId": 1, "quantity": -100 }
}
```

Remediation:
Add server-side validation to reject non-positive quantities:

```
if (req.body.quantity < 1) {
  return res.status(400).json({ error: 'Quantity must be at least 1' });
}
```

2.11 JWT Token Information Leakage

Attribute	Detail
Severity	MEDIUM
Type	Sensitive Data Exposure
Location	JWT tokens issued by `POST /rest/user/login`

Description:
JWT tokens issued by the application embed sensitive user data in the payload, including the user's MD5 password hash, role, and profile image path. Since JWTs are base64-encoded (not encrypted), this data is readable by anyone who intercepts or inspects the token.

Business Impact:
Every authenticated request transmits the user's password hash in the clear. Combined with the use of weak MD5 hashing, this means any network observer, browser extension, or XSS payload can extract the hash and crack the password offline.

Evidence:
Decoded JWT payload:

```
{
  "data": {
    "id": 1,
    "email": "admin@juice-sh.op",
    "password": "0192023a7bbd73250516f069df18b500",
    "role": "admin"
  }
}
```

Remediation:

1. Remove sensitive fields (especially `password`) from the JWT payload. Only include the minimum necessary claims (`sub` /user ID, `role`, `iat`, `exp`).
2. Implement token expiration (`exp` claim) -- currently tokens appear to have no expiry.
3. Store tokens in HttpOnly, Secure cookies instead of localStorage to reduce XSS-based theft.

2.12 User Enumeration via Admin API

Attribute	Detail
Severity	MEDIUM
Type	Broken Access Control / Information Disclosure
Location	`GET /api/Users`

Description:
Once an admin token is obtained (via SQL injection or mass assignment), the Users API returns the complete user database, including emails, roles, deluxe membership tokens, and account status for all 31 registered users.

Business Impact:
The entire customer database can be exfiltrated in a single request. This enables targeted phishing, credential stuffing against other services, and identification of high-value admin/deluxe accounts for further attack.

Evidence (sample):

ID	Email	Role
1	admin@juice-sh.op	admin
4	bjoern.kimminich@gmail.com	admin
5	ciso@juice-sh.op	deluxe
6	support@juice-sh.op	admin

Total: 31 users (6 admins, 3 deluxe, 22 customers).

Remediation:
Restrict the `/api/Users` endpoint to only return the currently authenticated user's data. If admin user listing is required, implement it behind a separate, audit-logged administrative interface with additional authentication controls (e.g., MFA).

2.13 Metrics Endpoint Information Disclosure

Attribute	Detail
Severity	LOW
Type	Information Disclosure / Security Misconfiguration
Location	`GET /metrics`

Description:
A Prometheus metrics endpoint is publicly accessible without authentication, exposing internal application statistics including HTTP request counts by status code, file upload statistics by type, and process-level CPU/memory metrics.

Business Impact:
Attackers can use this data to understand application usage patterns, identify successful attack indicators (e.g., 5XX error spikes), and discover additional attack surfaces (e.g., the 4 failed XML upload attempts suggest XXE testing opportunities).

Evidence (sample):

```
file_uploads_count{file_type="application/pdf"} 2
file_upload_errors{file_type="application/xml"} 4
http_requests_count{status_code="5XX"} 381
```

Remediation:
Restrict the `/metrics` endpoint to internal networks or authenticated monitoring systems only. Configure firewall rules or middleware to block public access.

---

3. Additional Security Weaknesses

The following security weaknesses were identified during reconnaissance but were **not directly exploited** during the attack phase. They warrant further investigation:

Missing Security Headers

- **Content-Security-Policy (CSP):** Not implemented -- would mitigate XSS impact.
- **Strict-Transport-Security (HSTS):** Missing -- no HTTPS enforcement.
- **X-XSS-Protection:** Absent.
- **Referrer-Policy:** Not configured.

Overly Permissive CORS

- `Access-Control-Allow-Origin: *` is set on all responses, meaning any external website can make authenticated cross-origin requests. Combined with XSS, this dramatically amplifies token theft risk.

No Rate Limiting

- Login, password reset, search, and captcha endpoints showed no rate limiting during testing. This enables brute-force attacks and automated abuse.

Captcha Answer in Response

- The `GET /rest/captcha/` endpoint returns the answer alongside the challenge (e.g., `{"captcha": "1-4*10", "answer": "-39"}`), completely defeating its anti-automation purpose.

Potential XXE in B2B API

- The B2B API (`/b2b/v2/orders`) documentation references deprecated XML endpoints. Metrics show 4 failed XML upload attempts. XML parsing may still be reachable through alternative request paths.

WebSocket Endpoint

- Socket.io is exposed at `/socket.io/` without visible authentication. This may allow real-time message injection or event manipulation.

Weak Cryptography

- All observed password hashes use unsalted MD5, which is considered cryptographically broken. Hashes can be cracked in seconds using precomputed rainbow tables.

Open Redirect

- The `/redirect?to=` endpoint performs URL validation but may accept whitelisted or relative URLs that could be abused for phishing.

---

4. Remediation Priority Matrix

Priority	Vulnerability	Severity	Effort
P1	SQL Injection -- Login Auth Bypass	Critical	Medium
P1	Password Hash Exposure via /rest/memories	Critical	Low
P1	Mass Assignment -- Admin Registration	Critical	Low
P2	IDOR -- Shopping Basket Access	High	Medium
P2	Stored XSS -- Product Reviews	High	Medium
P2	Directory Listing & File Exposure	High	Low
P2	Public Admin Configuration Endpoint	High	Low
P2	SQL Injection -- Product Search	High	Medium
P3	Null Byte Injection -- File Bypass	Medium	Low
P3	Business Logic -- Negative Quantities	Medium	Low
P3	JWT Sensitive Data in Payload	Medium	Medium
P3	User Enumeration via API	Medium	Low
P4	Metrics Information Disclosure	Low	Low

P4	Missing Security Headers (CSP, HSTS)	Low	Low
P4	CORS Wildcard Configuration	Low	Low
P4	No Rate Limiting	Low	Medium

---

5. Conclusion

The application is in a **critically vulnerable state**. Multiple independent attack paths lead to full administrative compromise without requiring any prior credentials. The combination of SQL injection, public password hash exposure, and mass assignment means that an attacker has at least three distinct methods to achieve admin access -- patching only one leaves the application exposed.

The most urgent action is to address the three critical vulnerabilities (SQL injection in login, password hash exposure, and mass assignment), followed by implementing proper authorization checks across all API endpoints. The use of MD5 for password hashing and the inclusion of password hashes in JWT tokens represent fundamental architectural issues that require a coordinated remediation effort including a forced password reset for all users.

Until these issues are resolved, the application should be considered fully compromised and unsuitable for production use with real user data.

Report prepared following penetration testing of http://192.168.86.249:3000 conducted on 2026-02-18.