**Penetration Testing Report for OWASP Juice Shop**

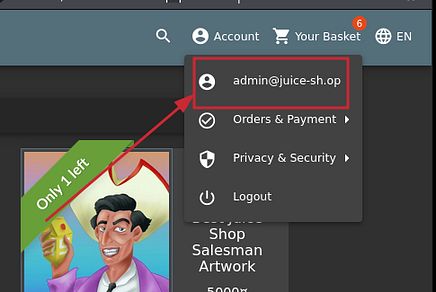
Presented by: Kushagra Pandurang Pitre

Date: 05/08/2024

I decided to check OWASP Juice Shop today. I will be writing about all the vulnerabilities and security issues I encounter, starting with testing the login functionality. This write-up will be the first, and I will indicate this in the title.

I tried using ' OR 1=1-- as the email and a random password, and it logged me into the admin account.

* **First vulnerability: Login is vulnerable to SQL injection.**



I clicked on the review section and noticed some users had left reviews. I thought, “What if I can log in as these users using SQL injection?”

**Recommendation :** Use parameterized queries to prevent SQL injection in the login functionality.

A screenshot of a video chat

Description automatically generated

I tried the following email addresses:

* bender@juice-sh.op
* stan@juice-sh.op

However, instead of logging in as those users, it logged me into the admin account.

* **Second vulnerability: Some User login is also vulnerable to SQL injection, but it grants admin access instead of the targeted user account.**

**Recommendation :** Implement input validation and use parameterized queries to prevent SQL injection that can grant admin access.

A screenshot of a computer

Description automatically generated

I created a test account: test+1@gmail.com. I noticed that the system allows a user to use their email as their password and also as the answer to the security question. This should not be allowed for security reasons.

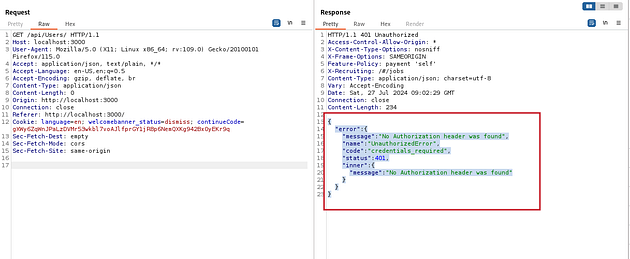
A screenshot of a computer

Description automatically generated

* **Third vulnerability: The system permits using the same email address for both the username and password, compromising account security.**

I went back to Burp Suite and noticed a POST request to /api/Users/. Curious, I sent it to the Repeater, removed the POST body, changed the request to GET, and sent it. I received a response error:

{  
 "error": {  
 "message": "No Authorization header was found",  
 "name": "UnauthorizedError",  
 "code": "credentials\_required",  
 "status": 401,  
 "inner": {  
 "message": "No Authorization header was found"  
 }  
 }  
}



I checked the site map for any requests with an Authorization header and found one. I decoded it and discovered it was the admin authorization token. I added it to the request, hoping it would work, and it did. I received users’ emails and roles in the response. I can log in using SQL injection without needing a password.

**Recommendation :** Enforce distinct and strong passwords that differ from the email address to enhance account security.

A screenshot of a computer

Description automatically generated

A computer code with black text

Description automatically generated

* **Fourth vulnerability: PII disclosure due to improper authorization checks.**

*Note: Normal user authorization tokens also worked.*

I made a request to /api/ after removing /Users/ from the endpoint, which triggered a stack error. I consider this a security issue.

**Recommendation :** Implement strict authorization checks to ensure only authorized users can access Personally Identifiable Information (PII).

A screenshot of a computer

Description automatically generated

* **Fifth vulnerability: Stack trace disclosure due to improper error handling.**

I noticed a header GET /rest/products/search?q= and tried apple' OR 1=1. This triggered an error that revealed the database name and the location of the error.

A screenshot of a computer

Description automatically generated

{  
 "error": {  
 "message": "SQLITE\_ERROR: near \"' OR 1=1%'\": syntax error",  
 "stack": "Error: SQLITE\_ERROR: near \"' OR 1=1%'\": syntax error",  
 "errno": 1,  
 "code": "SQLITE\_ERROR",  
 "sql": "SELECT \* FROM Products WHERE ((name LIKE '%apple' OR 1=1%' OR description LIKE '%apple' OR 1=1%') AND deletedAt IS NULL) ORDER BY name"  
 }  
}

**Recommendation :** Implement proper error handling to prevent stack trace disclosures in user-facing responses.

* **Sixth vulnerability:** **Information disclosure due to improper handling of SQL errors.**

I triggered another error via the /api/BasketItems/ endpoint by inputting a value it didn't expect (e.g., -1). This revealed more information about the database.

A screenshot of a computer screen

Description automatically generated

**Recommendation:** Sanitize SQL error messages to prevent information disclosure.

* **Seventh vulnerability: Information disclosure due to improper validation of user input.**

I noticed that a normal user can’t view the users who liked a particular product. By changing the authorization header to that of an admin, I was able to get the emails of users who liked the product.

A screenshot of a computer

Description automatically generated

**Recommendation:** Implement robust input validation to prevent information disclosure.

* **Eighth vulnerability: Privilege escalation allowing unauthorized access to user emails.**

**Recommendation:** Implement strict access controls to prevent privilege escalation and unauthorized access to user emails

**Conclusion**

Penetration testing revealed several critical and high-severity vulnerabilities in the OWASP Juice Shop application. Addressing these vulnerabilities should be a priority to improve application security. Implementing the given recommendations will significantly improve your overall security posture and protect you from potential attacks.