Vulnerability Assessment

&

Penetration Testing

Report prepared for

OWASP Juice Shop

Report issued :12/02/2026

Submitted By: Hari Krishnan V S

Confidentiality Statement

This document contains confidential security assessment information. Unauthorized distribution, copying, or disclosure of this report is strictly prohibited. The findings are intended solely for educational and security improvement purposes.

Disclaimer

This assessment was conducted as a “point-in-time” evaluation of the web application(s) in scope. While every effort has been made to identify vulnerabilities, this report may not disclose all existing security issues. Vulnerabilities identified in this report are based on the environment and data provided during the assessment period. Any changes to the systems or environment may affect the validity of the findings.

Table of Contents

1. Abstract

This report presents a comprehensive security assessment of the OWASP Juice Shop web application an intentionally vulnerable application designed for security training and awareness. The assessment was conducted to identify, analyze, and document security vulnerabilities that could compromise the application's confidentiality, integrity, and availability.

The evaluation employed a systematic methodology combining automated scanning tools and manual penetration testing techniques. Key tools utilized include Burp Suite Professional for comprehensive web application testing, SQLmap for automated SQL injection detection, and Dirsearch for directory enumeration. The assessment uncovered several critical and high-severity vulnerabilities across multiple attack vectors.

Critical vulnerabilities identified include SQL injection flaws allowing authentication bypass, reflected Cross-Site Scripting (XSS) enabling malicious script execution, exposed FTP directories revealing sensitive system information, and privilege escalation vulnerabilities through insecure basket manipulation. These findings represent significant security risks that could lead to unauthorized access, data breaches, and complete system compromise.

This report provides detailed documentation of each vulnerability, including technical descriptions, proof-of-concept demonstrations, impact assessments, and comprehensive remediation strategies. The findings underscore the critical importance of implementing secure coding practices, input validation, proper access controls, and regular security assessments to protect web applications against evolving cyber threats.

2. Introduction

In today's interconnected digital landscape, web applications serve as the backbone of modern business operations, facilitating everything from e-commerce transactions to critical infrastructure management. However, this increased reliance on web technologies has made these applications attractive targets for cybercriminals seeking to exploit vulnerabilities for financial gain, data theft, or malicious disruption.

The OWASP Juice Shop represents a modern single-page application built with JavaScript technologies, deliberately incorporating security vulnerabilities commonly found in real-world web applications. While designed as a security training platform, the vulnerabilities present in Juice Shop mirror those frequently discovered in production environments, making this assessment highly relevant for understanding contemporary web application security challenges.

This penetration testing engagement was conducted to systematically identify and document security weaknesses within the OWASP Juice Shop application. The primary objectives included discovering authentication bypass mechanisms, identifying injection vulnerabilities, uncovering information disclosure issues, and assessing authorization controls. The assessment followed industry-standard methodologies aligned with OWASP Testing Guide and PTES (Penetration Testing Execution Standard) frameworks.

The findings presented in this report serve multiple purposes: they demonstrate the potential impact of common web application vulnerabilities, provide technical details enabling informed remediation decisions, and highlight the importance of implementing defense-in-depth security strategies. By addressing these vulnerabilities, organizations can significantly reduce their attack surface and protect sensitive user data from compromise.

3. Executive Summary

The purpose of this assessment was to conduct a comprehensive security evaluation of the OWASP Juice Shop web application to uncover potential vulnerabilities and assess their impact on the application's security posture.

Key Findings

The assessment identified 4 vulnerabilities, categorized by severity:

- Critical: 2
- High: 2
- Medium: 0

The critical vulnerabilities pose significant risks, including unauthorized access to sensitive data, complete authentication bypass, execution of malicious scripts in user browsers, and potential for complete system compromise.

Recommendations

It is imperative to address the identified vulnerabilities immediately. Implementing the recommended remediations will significantly mitigate potential risks and enhance the overall security posture of the application. Priority should be given to the two critical vulnerabilities (SQL Injection and XSS) as they present immediate threats to user data and application integrity.

4. Scope

This security assessment focused on identifying security loopholes within the scope defined below. No additional information beyond the target application URL was provided. The assessment began with zero knowledge assumptions and followed a black-box testing approach.

In-Scope Items

- Target Application: http://192.168.1.11:3000/
- Application Type: OWASP Juice Shop Web Application
- Testing Window: December 2024
- Testing Type: Black-box penetration testing

Testing Areas Covered

- Authentication and Session Management
- Input Validation and Injection Vulnerabilities
- Authorization and Access Control
- Information Disclosure
- Business Logic Flaws

5. Testing Methodology

This assessment was carried out using a systematic and industry-standard approach to ensure comprehensive identification of security vulnerabilities. The methodology consisted of the following phases

Phase 1: Information Gathering and Reconnaissance

Initial reconnaissance was performed to understand the application structure and identify potential attack surfaces:

- Application mapping and crawling
- Technology stack identification
- Directory and file enumeration
- Endpoint discovery

Phase 2: Vulnerability Assessment

Automated and manual testing techniques were employed to identify vulnerabilities:

- SQL Injection testing using SQLmap and manual payloads
- Cross-Site Scripting (XSS) vulnerability scanning
- Authentication bypass attempts
- Authorization testing and privilege escalation
- Business logic flaw identification

Phase 3: Exploitation and Validation

Identified vulnerabilities were validated through controlled exploitation to confirm their exploitability and assess their potential impact. All testing was performed in a controlled manner to avoid disruption to the application.

Tools Utilized

- Burp Suite Professional: Comprehensive web application security testing
- SQLmap: Automated SQL injection detection and exploitation
- Dirsearch: Directory and file enumeration
- Browser Developer Tools: Client-side vulnerability analysis

6. Risk Classification

The vulnerabilities identified during the assessment were prioritized based on their severity using the CVSS (Common Vulnerability Scoring System). This classification ensures consistency in evaluating the potential impact and urgency of the vulnerabilities.

Critical (CVSS Score: 9.0 – 10.0)

Critical vulnerabilities represent the highest level of risk, often leading to catastrophic consequences if exploited. These vulnerabilities typically allow complete system compromise, unauthorized access to all data, or the ability to execute arbitrary code. Immediate action is required to mitigate these risks.

High (CVSS Score: 7.0 – 8.9)

High-risk vulnerabilities are exploitable with relative ease and can cause significant damage to the application, its data, or its users. They should be addressed promptly as part of an urgent remediation plan.

Medium (CVSS Score: 4.0 – 6.9)

Medium-risk vulnerabilities require specific conditions for exploitation and pose a moderate threat. Resolving these issues should be part of a planned security strategy and addressed in the near term.

Low (CVSS Score: 0.1 – 3.9)

Low-risk vulnerabilities have limited impact but still pose potential security concerns. They can be addressed during routine maintenance cycles or as part of general security improvements.

CVSS Score	Severity Level	Description
9.0 – 10.0	Critical	Immediate action required to mitigate risks.
7.0 – 8.9	High	High-priority vulnerabilities with significant risks.
4.0 – 6.9	Medium	Moderate vulnerabilities requiring resolution.
0.1 – 3.9	Low	Minor issues with minimal impact.

7. Assessment Findings

The following table summarizes all vulnerabilities identified during the assessment:

No	Finding	CVSS	Severity
1	SQL INJECTION	9.8	**Critical**
2	CROSS-SITE SCRIPTING (XSS)	9.0	**Critical**
3	EXPOSED FTP DIRECTORY	7.5	**High**
4	PRIVILEGE ESCALATION VIA BASKET MANIPULATION	8.0	**High**

Vulnerability #1: SQL Injection

Severity: **Critical (CVSS 9.8)**

Vulnerable URL: http://192.168.1.11:3000/#/login

Security Impact: **Severe**

Description

SQL Injection is a critical web security vulnerability that allows an attacker to interfere with the queries an application makes to its database. This vulnerability occurs when user input is improperly sanitized or validated and is directly included in SQL queries without proper parameterization. In the OWASP Juice Shop application, the login form is vulnerable to SQL injection, allowing attackers to bypass authentication mechanisms entirely by manipulating the SQL query structure. By injecting malicious SQL code into the email field, an attacker can create a query that always evaluates to true, granting unauthorized access to user accounts without valid credentials.

Impact

- Complete authentication bypass allowing unauthorized access to any user account
- Unauthorized access to sensitive customer data including personal information, order history, and payment details
- Potential for data modification or deletion within the database
- Administrative account compromise leading to complete system takeover
- Potential for lateral movement to other systems if database credentials are reused

Steps to Reproduce

1. Navigate to the OWASP Juice Shop login page: http://192.168.1.11:3000/#/login

2. In the Email field, enter the following SQL injection payload: ' OR 1=1#

3. In the Password field, enter any arbitrary value (e.g., 'password')

4. Click the 'Log in' button

5. Observe successful authentication and access to the admin account without providing valid credentials

Technical Details

The SQL injection payload ' OR 1=1# works by: • The single quote (') closes the original email string in the SQL query • OR 1=1 creates a condition that always evaluates to true • The hash symbol (#) comments out the rest of the original query, including the password check This results in a modified query that bypasses authentication entirely, returning the first user in the database (typically the administrator account).

Remediation

1. Implement Parameterized Queries (Prepared Statements)

The most effective defense against SQL injection is using parameterized queries or prepared statements. This ensures all user inputs are treated as data, not executable code. Instead of concatenating user input into SQL queries, use placeholders that the database engine properly escapes.

2. Input Validation and Sanitization

Implement comprehensive input validation on both client and server sides. Validate all user inputs for length, type, format, and range before processing. Use allowlists to define acceptable input patterns and reject anything that doesn't conform. For email fields specifically, validate against standard email format regex patterns.

3. Use ORM Frameworks

Utilize Object-Relational Mapping (ORM) frameworks that abstract database queries and automatically use parameterized queries. Modern ORMs like Sequelize (for Node.js) provide built-in protection against SQL injection when used correctly.

4. Apply Least Privilege Principle

Configure database accounts with minimal necessary permissions. The application should use database accounts that cannot perform administrative operations. Separate read-only operations from write operations using different database users.

5. Implement Web Application Firewall (WAF)

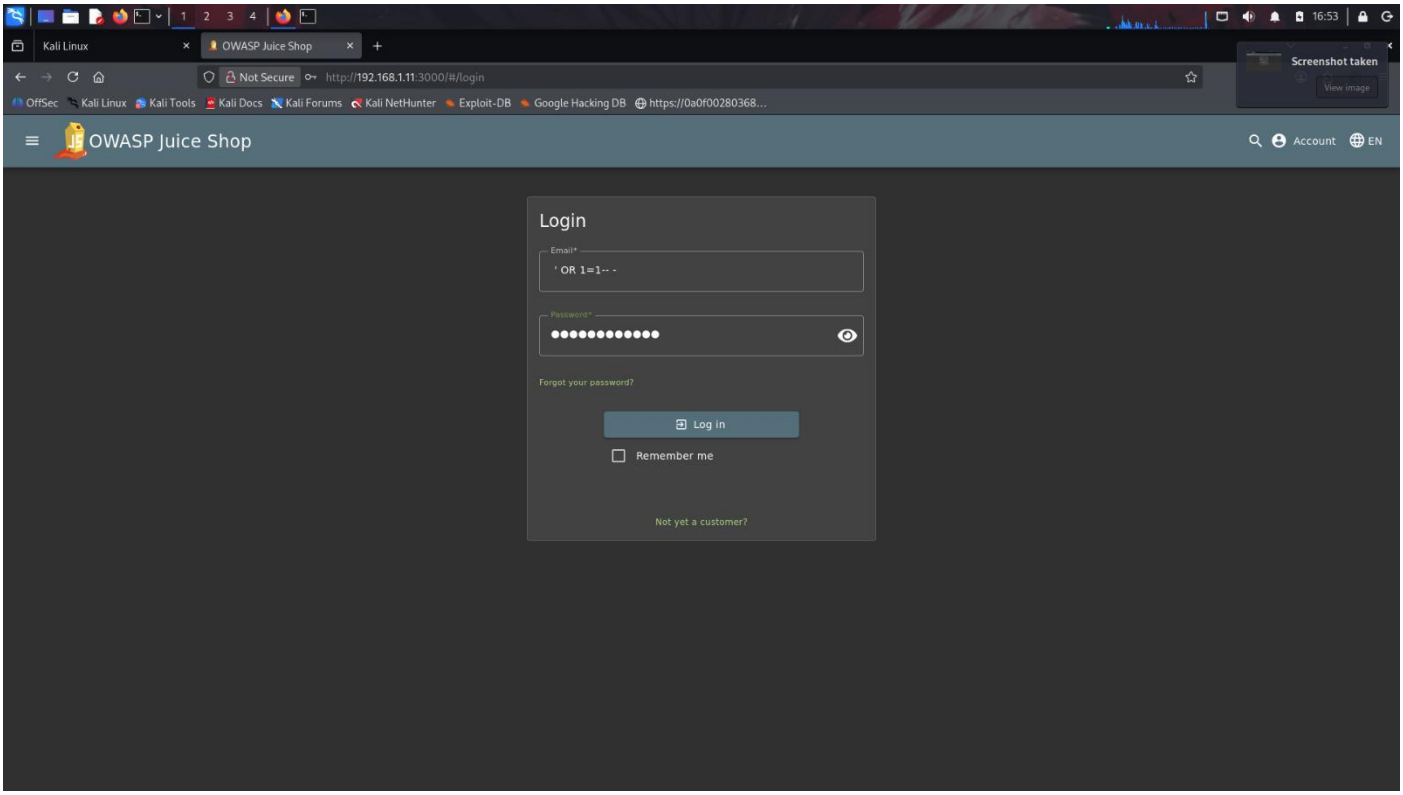
Deploy a Web Application Firewall configured to detect and block SQL injection attacks. Configure the WAF to analyze requests for malicious SQL patterns and automatically block suspicious requests.

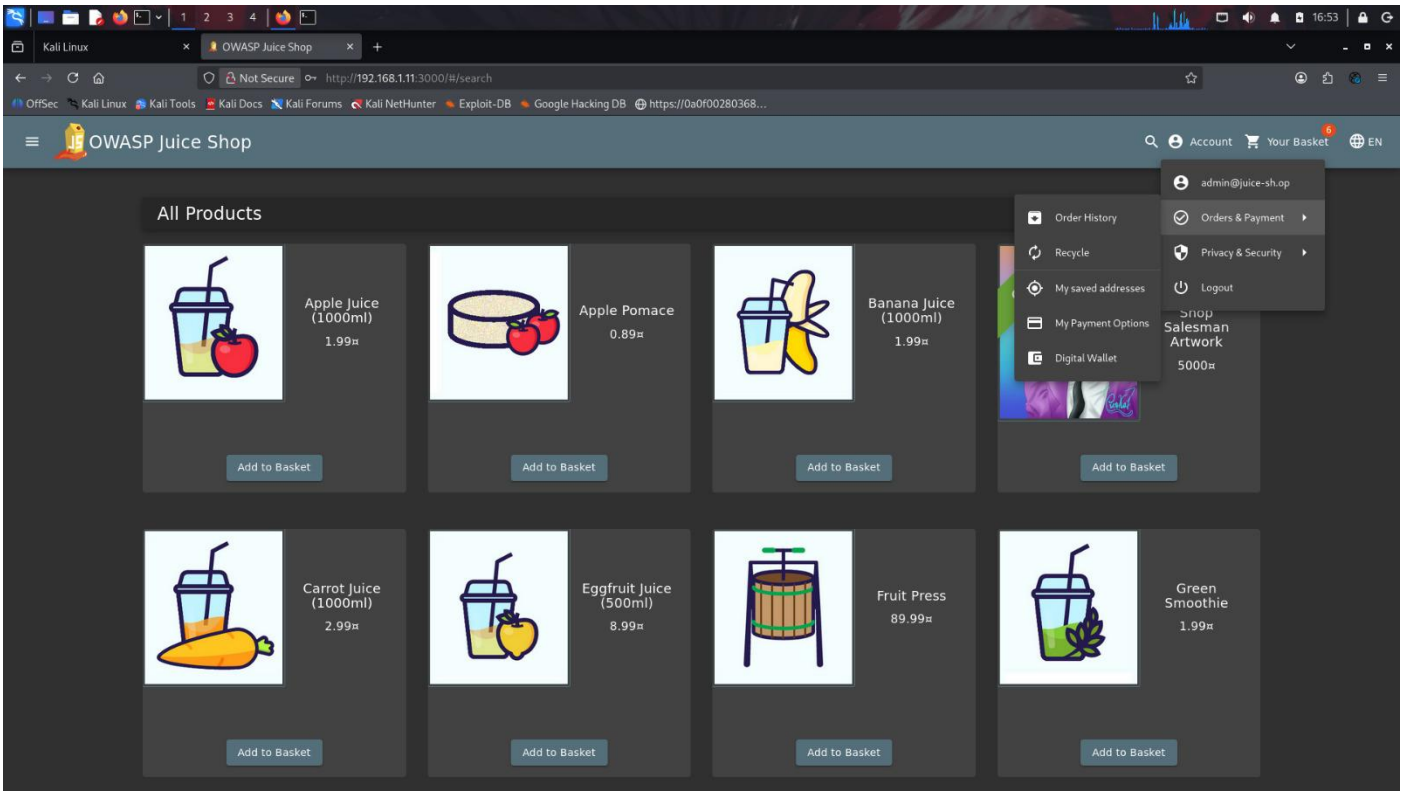
6. Regular Security Audits

Conduct regular code reviews and penetration testing to identify SQL injection vulnerabilities. Implement automated security scanning as part of the CI/CD pipeline.

References

- OWASP SQL Injection: https://owasp.org/www-community/attacks/SQL_Injection
- OWASP Top 10 A03:2021 - Injection: https://owasp.org/Top10/A03_2021-Injection/
- CWE-89: SQL Injection: https://cwe.mitre.org/data/definitions/89.html


Kali Linux
OWASP Juice Shop
Not Secure
http://192.168.1.11:3000/#/login
Screenshot taken
View image
OWASP Juice Shop
Account
EN
Login
Email*
' OR 1=1-- -
Password*
Forgot your password?
Log in
Remember me
Not yet a customer?


Kali Linux
OWASP Juice Shop
Not Secure
http://192.168.1.11:3000/#/search
OWASP Juice Shop
Account
Your Basket
EN
admin@juice-sh.op
Order History
Orders & Payment
Recycle
Privacy & Security
My saved addresses
Logout
My Payment Options
Digital Wallet
All Products
Apple Juice (1000ml)
1.99¤
Apple Pomace
0.89¤
Banana Juice (1000ml)
1.99¤
Salesman Artwork
5000¤
Carrot Juice (1000ml)
2.99¤
Eggfruit Juice (500ml)
8.99¤
Fruit Press
89.99¤
Green Smoothie
1.99¤
Add to Basket

Vulnerability #2: Cross-Site Scripting (XSS)

Severity: **Critical (CVSS 9.0)**

Vulnerable URL: http://192.168.1.11:3000/#/search

Security Impact: **Severe**

Description

Cross-Site Scripting (XSS) is a critical web security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. The OWASP Juice Shop search functionality is vulnerable to reflected XSS, where user-supplied input is directly rendered in the browser without proper sanitization or encoding. This vulnerability enables attackers to execute arbitrary JavaScript code in the context of a victim's browser session, potentially compromising user data, session tokens, and performing unauthorized actions on behalf of the victim.

Impact

- Session hijacking through theft of authentication cookies and tokens
- Account takeover by capturing user credentials through fake login forms
- Defacement of the website for affected users
- Redirection to malicious websites or phishing pages
- Keylogging and capture of sensitive information entered by users
- Execution of unauthorized actions using the victim's authenticated session

Steps to Reproduce

1. Navigate to the OWASP Juice Shop application

2. Locate the search bar in the top navigation

3. Enter the following XSS payload: <iframe src="javascript:alert('xss')">

4. Press Enter or click the search button

5. Observe the JavaScript alert dialog displaying 'xss', confirming script execution

Alternative XSS Payloads

Additional XSS payloads that successfully execute in this vulnerability: • <script>alert('XSS')</script> • <img src=x onerror=alert('XSS')> • <svg onload=alert('XSS')>

Remediation

1. Output Encoding and Escaping

Implement context-aware output encoding for all user-supplied data before rendering it in HTML. Use appropriate encoding functions based on the context (HTML entity encoding, JavaScript encoding, URL encoding, or CSS encoding). Modern frameworks like Angular provide automatic escaping but ensure its enabled and not bypassed.

2. Content Security Policy (CSP)

Implement a strict Content Security Policy that restricts the sources from which scripts can be loaded and prevents inline script execution. A properly configured CSP significantly reduces the impact of XSS vulnerabilities by preventing execution of injected scripts.

3. Input Validation

Validate all user inputs on the server side. Implement allowlists defining acceptable input patterns and reject anything that doesn't conform. While input validation alone is not sufficient to prevent XSS, it provides an important layer of defense in depth.

4. Use Security Headers

Implement security headers including X-XSS-Protection, X-Content-Type-Options, and a strict Content-Security-Policy. These headers provide additional browser-level protection against XSS attacks.

5. Framework Security Features

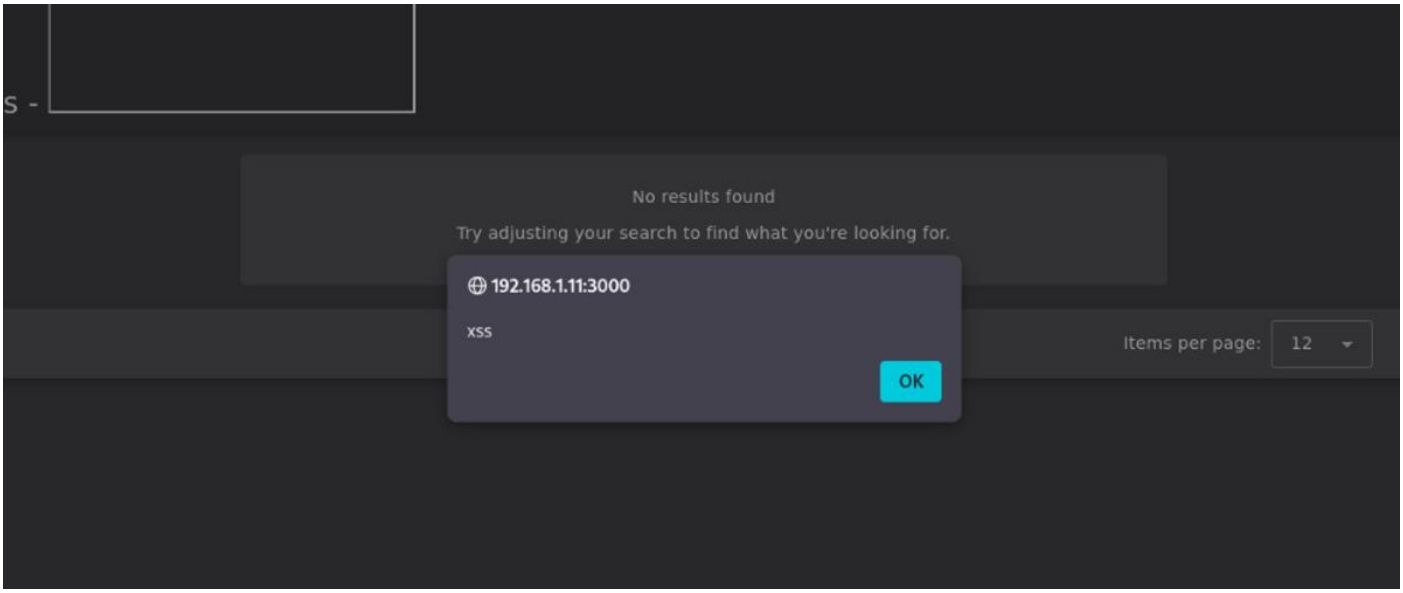
Leverage built-in security features of modern frameworks. Angular provides automatic sanitization of values, but developers must avoid bypassing these protections using methods like bypass Security Trust*. Ensure all developers understand proper use of framework security features.

6. Regular Security Testing

Conduct regular security testing including automated scanning and manual penetration testing to identify XSS vulnerabilities. Include XSS testing in the software development lifecycle and implement automated security testing in CI/CD pipelines.

References

- OWASP XSS: https://owasp.org/www-community/attacks/xss/
- OWASP Top 10 A03:2021 - Injection: https://owasp.org/Top10/A03_2021-Injection/
- CWE-79: Cross-site Scripting: https://cwe.mitre.org/data/definitions/79.html



Vulnerability #3: Exposed FTP Directory

Severity: High (CVSS 7.5)

Vulnerable URL: http://192.168.1.11:3000/ftp

Security Impact: High

Description

The OWASP Juice Shop application exposes an FTP directory with directory listing enabled, allowing unauthorized users to browse and potentially download sensitive files. This vulnerability represents a significant information disclosure issue where the web server is configured to display the contents of directories that should not be publicly accessible. The exposed directory contains various files including backup files, configuration files, and other sensitive documents that provide valuable information to potential attackers.

Impact

- Information disclosure revealing internal system structure and file organization
- Exposure of sensitive files including backups, configuration files, and documentation
- Potential disclosure of credentials or API keys stored in configuration files
- Intelligence gathering for targeted attacks based on exposed system information
- Possible access to source code or proprietary business information

Steps to Reproduce

1. Perform directory enumeration using Dirsearch or similar tools

2. Navigate directly to: http://192.168.1.11:3000/ftp

3. Observe the directory listing displaying various files

4. Files can be directly accessed and downloaded by clicking on them

Exposed Files Include

• Package backup files (.bak) • Configuration files • Legal documents • Support documentation • Acquisition-related files • Easter egg files

Remediation

1. Disable Directory Listing

Configure the web server to disable directory listing for all directories. For Apache servers, remove the 'Indexes' option from the Options directive. For Nginx, ensure autoindex is set to off. This prevents attackers from browsing directory contents even if they discover the directory path.

2. Implement Access Controls

Restrict access to sensitive directories using server-level access controls. Implement authentication requirements for accessing administrative or backup directories. Use .htaccess files (Apache) or location blocks (Nginx) to deny access to unauthorized users.

3. Move Sensitive Files Outside Webroot

Store sensitive files, backups, and configuration files outside the web server's document root. This ensures these files cannot be accessed via HTTP requests regardless of server configuration. If files must be served, implement controlled access through application logic.

4. Regular Security Audits

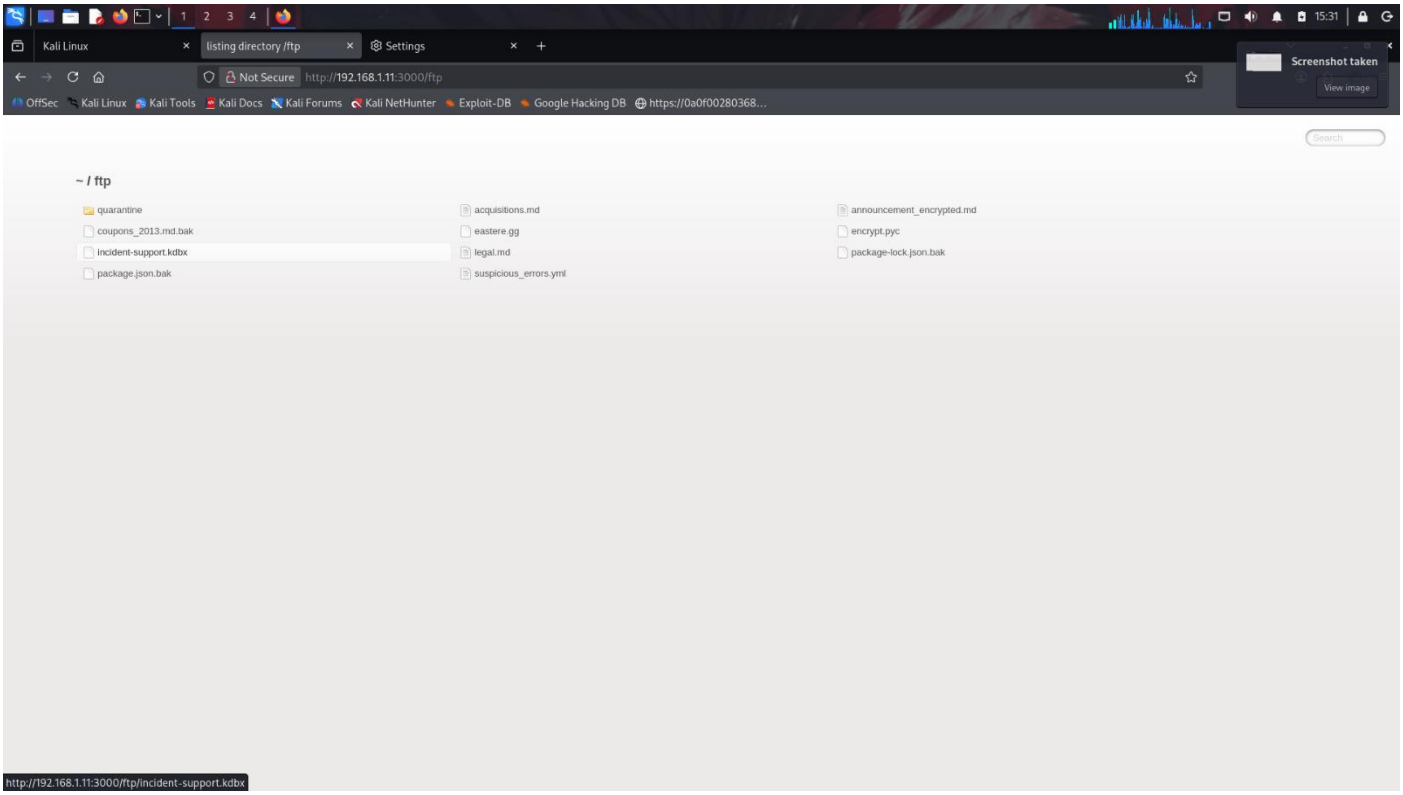
Conduct regular security audits to identify publicly accessible directories and files. Use automated scanning tools to detect directory listing vulnerabilities and misconfigured access controls. Review web server configurations as part of routine security maintenance.

5. Implement robots.txt and Security Headers

While not a security control, use robots.txt to discourage search engine indexing of sensitive directories. Implement security headers like X-Content-Type-Options and X-Frame-Options to provide additional protection layers.

References

- OWASP Information Disclosure: https://owasp.org/www-community/vulnerabilities/Information_exposure_through_directory_listing
- CWE-548: Directory Listing: https://cwe.mitre.org/data/definitions/548.html



Vulnerability #4: Privilege Escalation via Basket Manipulation

Severity: High (CVSS 8.0)

Vulnerable Endpoint: http://192.168.1.11:3000/rest/basket/*

Security Impact: High

Description

The OWASP Juice Shop application contains a critical authorization vulnerability in its basket functionality that allows users to access and manipulate other users' shopping baskets. The application fails to properly validate basket ownership, relying solely on a sequential basket ID passed in the request. An attacker can enumerate basket IDs and modify the basket parameter in API requests to access, view, or modify other users' shopping carts. This represents a fundamental failure in implementing proper authorization controls and demonstrates an Insecure Direct Object Reference (IDOR) vulnerability.

Impact

- Unauthorized access to other users' shopping baskets and order information
- Ability to view sensitive customer data including product selections and potentially payment information
- Manipulation of other users' orders by adding or removing items
- Privacy violation through exposure of customer purchasing behavior
- Potential for denial of service by deleting items from user baskets

Steps to Reproduce

1. Log into the OWASP Juice Shop application

2. Add items to your shopping basket

3. Open Burp Suite and configure the browser to use Burp as a proxy

4. Navigate to the basket page and intercept the request

5. In the intercepted request, locate the basket ID parameter (e.g., GET /rest/basket/2)

6. Modify the basket ID to a different number (e.g., change /basket/2 to /basket/1 or /basket/3)

7. Forward the modified request

8. Observe access to another user's basket contents, confirming the IDOR vulnerability

Technical Details

The vulnerability exists because the application: • Uses predictable, sequential basket identifiers • Fails to verify that the authenticated user owns the requested basket • Trusts client-supplied basket IDs without server-side validation • Does not implement proper session-basket binding checks

Remediation

1. Implement Proper Authorization Checks

Implement server-side authorization checks for every basket access request. Verify that the authenticated user owns the requested basket before returning any data or allowing modifications. Never rely solely on client-supplied identifiers for authorization decisions.

2. Use Indirect Reference Maps

Instead of using direct database IDs, implement an indirect reference map that creates session-specific, unpredictable identifiers for baskets. Map these session-specific identifiers to the actual database IDs server-side, ensuring users can only access their own resources.

3. Session-Based Basket Association

Associate baskets with user sessions server-side and retrieve basket information based on the authenticated session rather than accepting basket IDs from client requests. This eliminates the possibility of users manipulating identifiers to access other users' data.

4. Implement Access Control Lists

Maintain access control lists that explicitly define which users can access which resources. Before serving any basket data, verify that the requesting user appears in the access control list for that specific basket.

5. Use UUIDs Instead of Sequential IDs

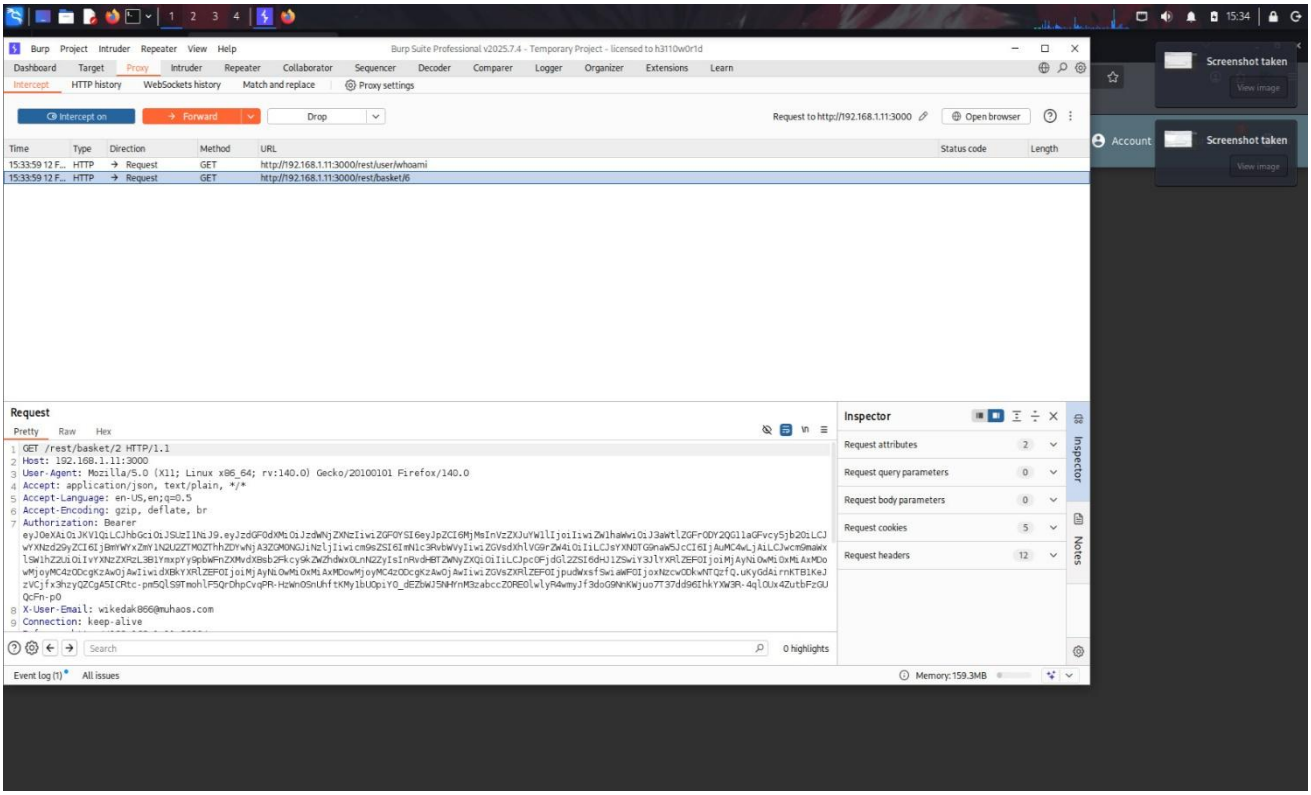
Replace sequential integer IDs with UUIDs (Universally Unique Identifiers) for basket references. While this doesn't eliminate the need for authorization checks, it significantly increases the difficulty of enumerating valid basket identifiers.

6. Regular Security Testing

Conduct regular security testing specifically focused on authorization vulnerabilities. Test all endpoints that accept resource identifiers to ensure proper ownership validation is enforced.

References

- OWASP IDOR: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/04-Testing_for_Insecure_Direct_Object_References
- OWASP Top 10 A01:2021 - Broken Access Control: https://owasp.org/Top10/A01_2021-Broken_Access_Control/
- CWE-639: Insecure Direct Object References: https://cwe.mitre.org/data/definitions/639.html



8. Appendix A: Tools Used

Tool	Description
Burp Suite Professional	Comprehensive web application security testing platform used for intercepting HTTP/HTTPS traffic, identifying vulnerabilities, and performing manual penetration testing
SQLmap	Automated SQL injection detection and exploitation tool used for testing database security
Dirsearch	Directory and file enumeration tool used for discovering hidden paths and exposed resources
Browser Developer Tools	Built-in browser tools for analyzing client-side code, network traffic, and JavaScript execution

9. Conclusion

This comprehensive security assessment of the OWASP Juice Shop web application has identified multiple critical and high-severity vulnerabilities that pose significant risks to the application's security posture. The assessment uncovered fundamental security weaknesses across several attack vectors, including injection vulnerabilities, authentication bypass, information disclosure, and broken access controls.

The two critical vulnerabilities—SQL Injection and Cross-Site Scripting—represent the most severe threats to the application. The SQL injection vulnerability allows complete authentication bypass, granting attackers unauthorized access to any user account, including administrative accounts. This vulnerability could lead to complete system compromise, unauthorized data access, and potential data manipulation or destruction. The XSS vulnerability enables attackers to execute malicious JavaScript in users' browsers, potentially leading to session hijacking, credential theft, and unauthorized actions performed on behalf of legitimate users.

The high-severity vulnerabilities—exposed FTP directory and privilege escalation through basket manipulation—while not immediately catastrophic, significantly increase the application's attack surface. The exposed FTP directory provides attackers with valuable intelligence about the system's internal structure and potentially sensitive files. The basket manipulation vulnerability demonstrates a fundamental failure in implementing proper authorization controls, allowing users to access and modify other users' data.

These findings underscore several critical security principles that must be implemented:

- **Input Validation and Output Encoding:** All user input must be properly validated and sanitized. Output must be encoded based on the context in which it is displayed.
- **Parameterized Queries:** Database interactions must use parameterized queries or prepared statements to prevent SQL injection attacks.
- **Proper Authorization Controls:** Server-side authorization checks must verify user permissions for every resource access request.
- **Principle of Least Privilege:** Systems and users should have only the minimum permissions necessary to perform their functions.
- **Defense in Depth:** Multiple layers of security controls should be implemented to protect against various attack vectors.

It is strongly recommended that the identified vulnerabilities be addressed as a matter of urgency, with priority given to the critical issues. The remediation strategies outlined in this report provide detailed guidance for

addressing each vulnerability. Additionally, implementing a comprehensive security program that includes secure coding training, regular security assessments, and automated security testing in the development pipeline will help prevent similar vulnerabilities from being introduced in the future.

The OWASP Juice Shop serves as an excellent learning platform that demonstrates real-world vulnerabilities and their potential impact. The vulnerabilities identified in this assessment are representative of those commonly found in production web applications. By understanding these vulnerabilities and their remediation strategies, development teams can build more secure applications and better protect their users' data and privacy.

This assessment demonstrates that comprehensive security testing is essential for identifying vulnerabilities before they can be exploited by malicious actors. Organizations should incorporate regular security assessments, penetration testing, and security code reviews as integral parts of their software development lifecycle to maintain a strong security posture and protect against evolving cyber threats.