

Dark Pyramid

Security Assessment Findings Report

Business Confidential

*Date: 25 JAN 2026
Project: Juice-Shop*

Confidentiality Statement

This document is the exclusive property of Dark Pyramid. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of Dark Pyramid.

Dark Pyramid may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Dark Pyramid prioritized the assessment to identify the weakest security controls an attacker would exploit. Dark Pyramid recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Contact Information

Name	Title	Contact Information
DarkPyramid		
Ibram Anwar	Founder&CTO@DarkPyramid	Email: Ibram@Dark-Pyramid.com
Sondos Sayed	Junior Penetration Tester	Email: sondosgaber98@gmail.com

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Risk Factors

Risk is measured by two factors: Likelihood and Impact:

Likelihood

Likelihood measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

Impact

Impact measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Reconnaissance Notes – OWASP Juice Shop Methodology

The assessment was conducted on the online version of OWASP Juice Shop (<https://juice-shop.herokuapp.com/>). The methodology included:

- 1- Manual Exploration: Navigating through the application to understand its functionality and identify key endpoints.
- 2- Endpoint Discovery: Identifying sensitive directories and pages such as /ftp, /administration, and /login.
- 3- Vulnerability Testing: Performing manual tests for common web vulnerabilities like SQL Injection (SQLi), Cross-Site Scripting (XSS), and Broken Access Control.
- 4- Request Analysis: Intercepting and analyzing HTTP requests and responses using browser developer tools and Burp Suite to identify hidden parameters, roles, and client-controlled data.
- 5- subdomains, URL, and Parameter Enumeration: Automated reconnaissance was performed to collect application-related assets, URLs, and parameters that could be leveraged for further security testing. This step focused on identifying accessible endpoints, parameters, and historical URLs that may not be immediately visible through normal browsing. The waymore tool was used during this phase to enumerate URLs, parameters, and related assets from publicly available sources using the following approach: (`waymore -i target_name -mode B -oU output_file`).
- 6- Vulnerability Identification: Based on the gathered reconnaissance data, manual testing was performed for common web application vulnerabilities, including SQL Injection (SQLi), Broken Access Control, Privilege Escalation, and Sensitive Data Exposure.

Tester Notes and Recommendations

During the security assessment conducted by Dark Pyramid, multiple critical and high-risk vulnerabilities were identified within the web application. These findings indicate weaknesses in authentication mechanisms, access control enforcement, and sensitive data handling.

- SQL Injection (Authentication Bypass) allowed attackers to bypass login controls and gain unauthorized access to user accounts, including privileged users.
- Privilege Escalation via Role Manipulation during Registration revealed improper trust in client-side input, enabling users to assign themselves administrative roles.
- Broken Access Control (IDOR) allowed unauthorized users to access other users' order information by directly manipulating object identifiers.
- Sensitive Data Exposure (Public FTP Directory) exposed internal and backup files, increasing the risk of information disclosure and further attacks.

Overall Risk and Recommendations

The identified vulnerabilities demonstrate a lack of robust server-side validation, insufficient authorization checks, and insecure handling of sensitive resources. If exploited together, these issues could lead to full application compromise, data leakage, and unauthorized administrative access.

Dark Pyramid recommends implementing strict server-side access control, enforcing proper role validation, securing sensitive directories, and conducting regular security testing to improve the overall security posture of the application.

Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

Penetration Test Findings

2	0	2	0	0
Critical	High	Medium	Low	Informational

Finding	Severity	Recommendation
001: SQL Injection (Authentication Bypass)	<u>Critical</u>	Use parameterized queries and strict input validation to prevent SQL injection attacks.
002: Privilege Escalation via Role Manipulation during Registration	<u>Critical</u>	Assign user roles only on the server side and never trust role values from client requests.
003: Broken Access Control – Viewing Other Users' Orders (IDOR)	<u>Medium</u>	Enforce server-side authorization checks to ensure users can access only their own data.
004: Sensitive Data Exposure (Public FTP Directory)	<u>Medium</u>	Disable directory listing and remove sensitive files from publicly accessible locations.

Technical Findings

001: SQL Injection (Authentication Bypass) (**Critical**)

Description:	This vulnerability allows an attacker to bypass the authentication mechanism entirely. By injecting SQL commands into the login form, an attacker can authenticate as any user, including the administrator, without knowing their password. This leads to a complete compromise of the application's data and user accounts.
Risk:	<p>Likelihood: High – The vulnerability is easily exploitable through the login form without requiring advanced skills or special access. Since the attack can be performed remotely using simple SQL payloads, it is highly likely to be abused by attackers, especially if the application is publicly accessible.</p> <p>Impact: Very High – An attacker can gain full administrative control over the application, access all user data, modify or delete records, and potentially use the administrative privileges to further compromise the system.</p>
Tools Used:	Web browser (manual testing)
References:	https://portswigger.net/web-security/sql-injection

Evidence

- 1- Open the browser and navigate to <https://juice-shop.herokuapp.com/#/login>.
- 2- In the "Email" field, type: ' OR 1=1 --
- 3- In the "Password" field, type any random string (e.g.,123).

- 4- Click the "Log in" button.
- 5- Observe that you are now logged in as the administrator (admin@juice-sh.op).

Suggested fix

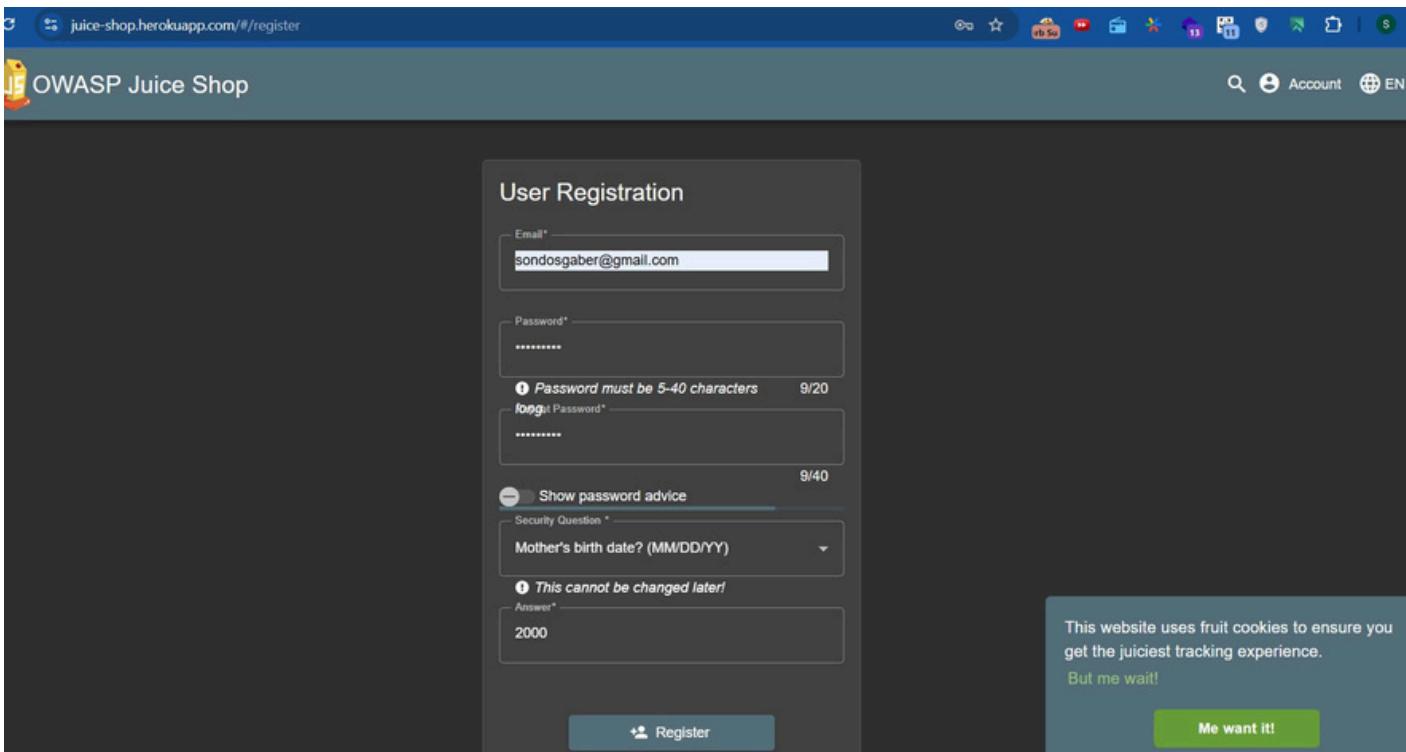
Use **parameterized queries** or **prepared statements** for all database interactions. This ensures that user input is treated as data rather than executable code. Additionally, implement a robust Web Application Firewall (WAF) to detect and block common SQL injection patterns.

002: Privilege Escalation via Role Manipulation during Registration (Critical)

Description:	The application allows users to modify the registration request and include a role parameter. By changing the value of this parameter to admin, an attacker can register a new account with administrative privileges. This issue occurs due to missing server-side validation and improper trust in user-supplied input during the registration process. As a result, a normal user can gain full administrative access without authorization.
Risk:	Likelihood: High – Exploitation is simple by intercepting and modifying the registration request (e.g., using Burp Suite). Impact: Very High - Full administrative access to the application.
Tools Used:	Web Browser, Burp Suite
References:	What Is Privilege Escalation? Definition, Types and Examples

Evidence

- 1- Navigate to the User Registration page of the application.
- 2- Start registering a new account like email: sondosgaber@gmail.com



The screenshot shows a browser window for the OWASP Juice Shop application at the URL juice-shop.herokuapp.com/#/register. The page title is "User Registration". The form fields are filled as follows:

- Email*: sondosgaber@gmail.com
- Password*: (obscured)
- Confirm Password*: (obscured)
- Security Question*: Mother's birth date? (MM/DD/YY)
- Answer*: 2000

A tooltip for the security question field states: "This cannot be changed later!" Below the form is a button labeled "Register". To the right of the form, there is a sidebar with the text: "This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wai!". At the bottom right of the sidebar is a green button labeled "Me want it!".

- 3- Check request and response of registering a new account in Burp Suite.

Request

Pretty Raw Hex

```

1 POST /api/Users/ HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomebanner_status=dismiss; continueCode
4 =axzjXDeL13J97PgBy60NBt9flYuMZF1kTxEcMvhKMOqQ82MoOinVmWRYW5EK
5 Content-Length: 260
6 Sec-Ch-Ua-Platform: "Windows"
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
8 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/144.0.0.0
9 Safari/537.36
10 Accept: application/json, text/plain, /*
11 X-User-Email: ' or l=1 --
12 Content-Type: application/json
13 Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144", "Google
14 Chrome";v="144"
15 Sec-Ch-Ua-Mobile: ?0
16 Origin: https://juice-shop.herokuapp.com
17 Sec-Fetch-Site: same-origin
18 Sec-Fetch-Mode: cors
19 Sec-Fetch-Dest: empty
20 Referer: https://juice-shop.herokuapp.com/
21 Accept-Encoding: gzip, deflate, br
22 Accept-Language: en-US,en;q=0.9,ar;q=0.8
23 Priority: u=1, i
24 Connection: keep-alive
25
26 {
27   "email": "sondosgaber@gmail.com",
28   "password": "Sondos123",
29   "passwordRepeat": "Sondos123",
30   "securityQuestion": {
31     "id": 3,
32     "question": "Mother's birth date? (MM/DD/YY)",
33     "createdAt": "2026-01-26T16:30:53.404Z",
34   }
35 }
```

Response

Pretty Raw Hex Render

```

1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 Content-Length: 312
4 Content-Type: application/json; charset=utf-8
5 Date: Mon, 26 Jan 2026 16:40:26 GMT
6 Etag: W/"138-YTMyyOl/SSQGbmACTBcscm4Tb34"
7 Feature-Policy: payment 'self'
8 Location: /api/Users/25
9 Nel:
10 ("report_to": "heroku-nel", "response_headers": ["Via"], "max_age": 3600, "success_fraction": 0.01, "failure_fraction": 0.1)
11 Report-To:
12 {"group": "heroku-nel", "endpoints": [{"url": "https://nel.herokuapp.com/reports?s=8c34Ms6R1iVPOLogEkq@qd1VbPVHjy6EE5f70YpQm4wt3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959\u0002&ts=1769445626"}], "max_age": 3600}
13 Reporting-Endpoints:
14 heroku-nel="https://nel.herokuapp.com/reports?s=8c34Ms6R1iVPOLogEkq@qd1VbPVHjy6EE5f70YpQm4wt3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&ts=1769445626"
15 Server: Heroku
16 Vary: Accept-Encoding
17 Via: 1.1 heroku-router
18 X-Content-Type-Options: nosniff
19 X-Frame-Options: SAMEORIGIN
20 X-Recruiting: /#/jobs
21
22 {
23   "status": "success",
24   "data": {
25     "username": "",
26     "role": "customer",
27     "deluxeToken": ""
28   }
29 }
```

4- Observe that the registration response contains a parameter such as: role=customer

5- Modify the registration request by adding the same parameter: role=admin and set new email: hacker@gmail.com

6- Forward the modified request to the server.

7- Observe that the account is created successfully.

Request

Pretty	Raw	Hex	Render
1 POST /api/Users/ HTTP/1.1			
2 Host: juice-shop.herokuapp.com			
3 Cookie: language=en; welcomebanner_status=dismiss; continueCode=axzjXDeL13j97PgBy6NBt9f1YuMZF1kTxEcMwhKMOqQ82MoOinVmRYW5EK			
4 Content-Length: 274			
5 Sec-Ch-Ua-Platform: "Windows"			
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/144.0.0.0 Safari/537.36			
7 Accept: application/json, text/plain, */*			
8 X-User-Email: ' or 1=1 --			
9 Content-Type: application/json			
10 Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144", "Google Chrome";v="144"			
11 Sec-Ch-Ua-Mobile: 20			
12 Origin: https://juice-shop.herokuapp.com			
13 Sec-Fetch-Site: same-origin			
14 Sec-Fetch-Mode: cors			
15 Sec-Fetch-Dest: empty			
16 Referer: https://juice-shop.herokuapp.com/			
17 Accept-Encoding: gzip, deflate, br			
18 Accept-Language: en-US,en;q=0.9,ar;q=0.8			
19 Priority: u=1, i			
20 Connection: keep-alive			
21			
22 {			
23 "email":"hacker@gmail.com",			
24 "role":"admin",			
25 "password":"Sondos123",			
26 "passwordRepeat":"Sondos123",			
27 "securityQuestion":{			
28 "id":3,			
29 "question":"Mother's birth date? (MM/DD/YY)",			
30 "createdAt":"2026-01-26T16:30:53.404Z",			
31 "updatedAt":"2026-01-26T16:30:53.404Z"			}

Response

Pretty	Raw	Hex	Render
1 "report_to": "heroku-nei", "response_headers": {"Via": "1.1 heroku-nei", "Content-Type": "application/json", "Content-Length": "3600", "Connection": "close", "Date": "Mon, 26 Feb 2024 16:30:53 GMT", "Server": "Heroku", "X-Content-Type-Options": "nosniff", "X-Frame-Options": "SAMEORIGIN", "X-Recruiting": "#/jobs", "X-Report-Endpoints": "https://heroku-nei.herokuapp.com/reports?s=ZOIIIdXpRLW393bvvq8XMXp4Q42TthHfhqAdjYRHNtcgkQt3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959\u0026ts=1769445676"}, "status": "success", "data": {			
2 "username": "",			
3 "deluxeToken": "",			
4 "lastLoginIp": "0.0.0.0",			
5 "profileImage": "/assets/public/images/uploads/defaultAdmin.png",			
6 "isActive": true,			
7 "id": 28,			
8 "email": "hacker@gmail.com",			
9 "role": "admin",			
10 "updatedAt": "2026-01-26T16:41:16.091Z",			
11 "createdAt": "2026-01-26T16:41:16.091Z",			
12 "deletedAt": null}			

8- Log in using the newly created account With email: hacker@gmail.com and password: Sondos123

The screenshot shows the OWASP Juice Shop login interface. The URL in the browser is juice-shop.herokuapp.com/#/login. The page features a dark background with a light gray header bar containing the site logo and navigation links. The main content area is titled "Login". It includes two input fields: "Email*" with the value "hacker@gmail.com" and "Password*" with the value "Sondos123". Below these fields are "Forgot your password?" and "Log in" buttons. A "Remember me" checkbox is checked. A horizontal line separates the login form from a "Log in with Google" button. At the bottom right, there is a note about fruit cookies and a link to "But me wait!".

9- Verify that the user now has administrative privileges, such as access to the admin dashboard.

10-Navigate to <https://juice-shop.herokuapp.com/#/administration>.

The screenshot shows the OWASP Juice Shop administration interface. On the left, there's a sidebar titled 'Administration' with a 'Registered Users' section containing a list of email addresses: admin@juice-sh.op, jim@juice-sh.op, bender@juice-sh.op, bjoern.kimminich@gmail.com, ciso@juice-sh.op, support@juice-sh.op, morty@juice-sh.op, mc.safesearch@juice-sh.op, and J12934@juice-sh.op. On the right, there's a 'Customer Feedback' section with several reviews. A user menu is open in the top right corner, showing options like 'Account', 'Orders & Payment', 'Privacy & Security', and 'Logout'. The 'Logout' option is highlighted with a yellow background. A tooltip at the bottom right says 'This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!'

11- Now hacker user become admin.

Suggested fix

- Enforce server-side role assignment:
The application must never accept or trust the role parameter from client-side requests during registration or profile updates. User roles should be assigned exclusively on the server.
- Implement strict input validation and whitelisting:
Validate incoming request parameters and explicitly reject any unauthorized or unexpected fields such as `role`.
- Apply proper access control checks:
Ensure that only authorized administrative users can modify user roles, and that this functionality is restricted to protected admin-only endpoints.
- Use secure defaults:
All newly registered users should be assigned the default role (e.g., customer) regardless of any client-supplied values.

003: Broken Access Control – Viewing Other Users' Orders (IDOR) ([Medium](#))

Description:	The application does not properly enforce access control on the basket endpoint. A regular authenticated user can manipulate the basket ID in the request and view orders belonging to other users.
Risk:	Likelihood: High – Requires only a valid user account, No special privileges needed. Impact: High – An attacker can access sensitive order information of other users, leading to unauthorized data exposure and violation of user privacy.
Tools Used:	Web Browser, Burp Suite
References:	Link

Evidence

- 1- Log in as a normal user.

The screenshot shows a web browser displaying the OWASP Juice Shop application at juice-shop.herokuapp.com/#/search. The page title is "All Products". There are three product cards visible: "Apple Juice (1000ml)" for 1.99€, "Apple Pomace" for 0.89€, and "Banana Juice (1000ml)" for 1.99€. Each card has an "Add to Basket" button. In the top right corner, there is a user menu for "sondos@sondos.com" which includes options for "Orders & Payment", "Privacy & Security", and "Logout".

- 2- Navigate to basket

A screenshot of a web browser displaying the OWASP Juice Shop website. The URL in the address bar is juice-shop.herokuapp.com/#/basket. The page title is "Your Basket" and it shows an email address (sondos@sondos.com) associated with the basket. Inside the basket, there is one item: "Apple Juice (1000ml)" with a small icon of a juice glass and an apple. To the right of the item are quantity controls (-, +, 1), the price "1.99", and a delete icon. Below the basket summary, the total price is displayed as "Total Price: 1.99". At the bottom of the page is a teal-colored button labeled "Checkout". A note below the button states "You will gain 0 Bonus Points from this order!". A footer banner at the bottom of the page reads "This website uses fruit cookies to ensure you".

3- Intercept the request using Burp Suite.

4- Observe a request similar to:

GET /rest/basket/6 HTTP/1.1

5- Modify the basket ID to another value related to another user :

GET /rest/basket/1 HTTP/1.1

6- Forward the request.

7- Observe that order details belonging to another user are returned.

Suggested fix

The application should enforce server-side authorization checks on all order-related endpoints. Specifically:

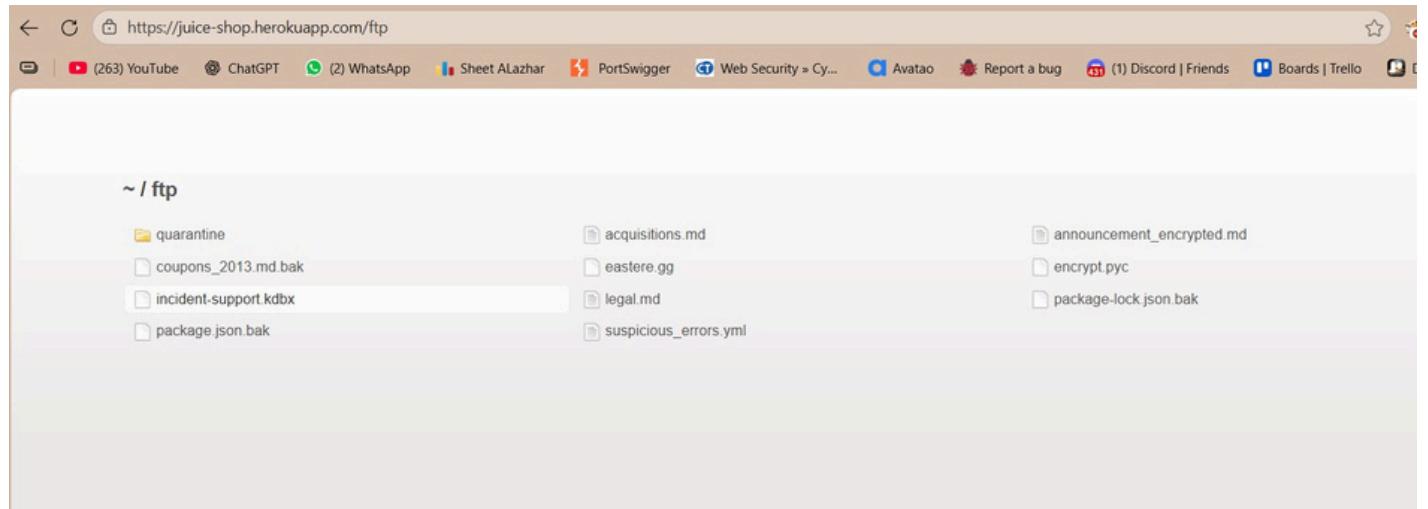
- Ensure that the backend verifies the authenticated user owns the requested order before returning any data.
- Avoid relying on client-side controls or hidden UI elements for access restriction.
- Implement proper access control logic (e.g., checking user_id against the order owner in the database).

004: Sensitive Data Exposure (Public FTP Directory) (Medium)

Description:	The application exposes a directory listing of the /ftp folder, which contains various files that should not be publicly accessible. This includes backup files and configuration files that can provide an attacker with valuable information about the system's architecture and potential secrets.
Risk:	<p>Likelihood: High – The exposed FTP directory is publicly accessible without authentication and requires no special tools or privileges. Any user who knows or guesses the /ftp path can access and download sensitive files easily.</p> <p>Impact: Medium – Attackers can gather information about the application's dependencies, internal structure, and potentially find hardcoded credentials or sensitive business logic in backup files.</p>
Tools Used:	Web Browser (Manual Testing)
References:	https://portswigger.net/web-security/information-disclosure

Evidence

- 1- Navigate to <https://juice-shop.herokuapp.com/ftp>.
- 2- Observe the directory listing showing files like coupons_2013.md.bak, package.json.bak, and incident-support.kdbx
- 3- Click on any file to download it.



Suggested fix

- Disable directory listing on the web server to prevent public file browsing.
- Remove backup and configuration files from publicly accessible directories.

- Store sensitive files outside the web root.
- Apply proper authentication and authorization controls if file access is required.
- Regularly review deployed files to ensure no sensitive artifacts are exposed.

If I were the defender

The primary goal of a defender is to reduce the attack surface and increase the cost and complexity for an attacker, while simultaneously improving the organization's ability to detect and respond to a breach. Based on common attack vectors and industry best practices, here are the proposed defensive measures.

5 Concrete Hardening Steps

System hardening is the process of securing a system by minimizing its attack surface and eliminating potential security gaps . The following five steps are concrete, actionable measures that would be immediately applied:

- 1- Disable Unnecessary Services and Ports: Perform a comprehensive audit of all running services and open network ports. Any service not essential for the system's core function (e.g., legacy protocols, unused administrative interfaces) must be disabled or uninstalled. This directly reduces the number of entry points an attacker can exploit.
- 2- Implement Principle of Least Privilege (PoLP): Review all user and service accounts to ensure they only possess the minimum permissions necessary to perform their required tasks. This includes restricting administrative access, removing default or generic accounts, and ensuring all applications run with the lowest possible privileges. This limits the lateral movement and damage an attacker can inflict upon compromise.
- 3- Enforce Strong Configuration Baselines: Apply standardized, secure configuration templates (e.g., CIS Benchmarks) to all operating systems, web servers (like Apache or Nginx), and applications. This involves measures such as enforcing strong password policies, configuring session timeouts, and disabling insecure features like directory browsing
- 4- Automate Patch and Vulnerability Management: Establish a rigorous, automated process for identifying, prioritizing, and applying security patches for all operating systems, firmware, and third-party software. Critical vulnerabilities must be patched within a defined, short service-level agreement (SLA), as unpatched systems are a primary vector for compromise.
- 5- Segment the Network and Implement Micro-segmentation: Divide the network into smaller, isolated zones based on function, risk, or data sensitivity. Crucially, implement micro-segmentation within data centers to enforce "zero-trust" principles, ensuring that communication between individual workloads is restricted to only what is absolutely necessary, thereby containing any breach .

One Monitoring/Logging Idea to Detect This Attack

Centralized Log Aggregation with Anomaly Detection on Authentication Events:
The most effective monitoring strategy would be to implement a Security Information and Event Management

(SIEM) system to centralize all logs—especially those related to authentication, authorization, and system configuration changes—from all critical assets. The specific idea to detect an attack is to:

- Monitor for Failed Authentication Spikes: Create a rule to alert on an unusually high volume of failed login attempts (e.g., more than 10 failed attempts from a single IP address or user account within a 60-second window). This is a classic indicator of brute-force or password-spraying attacks.
- Monitor for Successful Login from New Geolocation/IP: Establish a baseline of typical user login locations and alert on any successful login from a never-before-seen geographic location or a suspicious IP address (e.g., known VPN/proxy services or Tor exit nodes).
- Monitor for Privilege Escalation: Alert on any sequence of events that indicates a standard user account successfully executing a privilege escalation command or being added to an administrative group.

One Recommendation to Prevent Similar Issues in Future (Process/Tooling)

Implement a Continuous Vulnerability Management (CVM) Program Integrated with CI/CD:

To prevent similar issues from recurring, the organization must shift from reactive patching to a proactive, continuous security process. This involves two key components:

- 1- Process: Establish a formal Vulnerability Management Program that includes continuous scanning, risk-based prioritization, and clear ownership for remediation. This program must be championed by leadership and involve cross-functional teams (Development, Operations, Security).
- 2- Tooling: Integrate Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) tools directly into the Continuous Integration/Continuous Deployment (CI/CD) pipeline. This "shift-left" approach ensures that security vulnerabilities are identified and fixed by developers before the code is deployed to production, making remediation significantly cheaper and faster. The pipeline should be configured to automatically fail the build if a high-severity vulnerability is detected.