Pensar

ACME

Pentest Report

December 15, 2025

Table of Contents

1. Introduction

This document presents the findings and results of the penetration testing assessment conducted by Pensar for the coffee-shop application. The assessment was performed on December 15, 2025, and represents a comprehensive security evaluation of the application's attack surface and security controls.

The penetration testing engagement was conducted as a whitebox assessment, providing Pensar with full access to the application source code and running environment. This approach enables a thorough evaluation of both the application's external attack surface and internal security implementation details. The assessment focused on identifying security vulnerabilities, business logic flaws, and potential attack vectors that could be exploited by malicious actors.

This report is organized to provide both executive-level summaries and detailed technical findings. Security professionals and developers should review the complete findings section for remediation guidance, while executive stakeholders should focus on the Executive Summary and Conclusion sections for high-level risk assessment and business impact analysis.

2. Executive Summary

2.1 Purpose

Pensar performed a comprehensive penetration testing assessment of the coffee-shop application to identify security vulnerabilities, validate security controls, and provide actionable remediation guidance. The engagement was conducted as a whitebox assessment, allowing Pensar to evaluate both the application's external attack surface and internal code implementation.

The primary objectives of this assessment were to: (1) identify security vulnerabilities that could be exploited by attackers, (2) evaluate the effectiveness of existing security controls, (3) assess business logic implementation for potential abuse scenarios, and (4) provide specific, actionable recommendations for remediation and security improvement.

The testing methodology employed industry-standard penetration testing techniques, including reconnaissance and attack surface mapping, vulnerability scanning, manual exploitation testing, and business logic analysis. The assessment scope included one API endpoint and the associated backend application logic.

2.2 Findings Summary

Pensar identified a total of 2 security vulnerabilities during the assessment:

Finding Name	Risk Rating	Status
Missing Price Validation Allows Negative Product Prices	Critical	Open
Missing Validation for Negative Stock Quantities in Product Creation	High	Open

Critical Vulnerabilities 1	**High Vulnerabilities** 1
Medium Vulnerabilities 0	**Low Vulnerabilities** 0

2.3 Conclusion

The coffee-shop application contains critical security vulnerabilities that require immediate remediation. Specifically, the application fails to implement proper input validation for product pricing and inventory management, allowing authenticated administrators to create products with negative prices and stock quantities. These business logic vulnerabilities could be exploited to manipulate financial records, disrupt inventory management systems, and potentially facilitate fraud.

The presence of a critical vulnerability requires immediate action. Pensar strongly recommends prioritizing the remediation of the price validation vulnerability within 15 days, followed by the stock quantity validation issue within 30 days. Both vulnerabilities stem from insufficient input validation in the product creation and update endpoints and can be remediated through implementation of proper server-side validation logic.

While the assessment identified only two vulnerabilities, the critical nature of the price validation issue and its potential business impact warrant urgent attention. The application demonstrates some positive security practices, including JWT-based authentication and proper signature verification. However, the lack of comprehensive input validation on business-critical fields represents a significant security gap that must be addressed before the application is deployed to production or handles real financial transactions.

3. Scope

3.1 In Scope

The following assets and systems were included in the scope of this penetration testing assessment:

- **Application:** coffee-shop (Node.js/Next.js API application)
- **Repository:** joshkotrous/coffee-shop (main branch)
- **Assessment Type:** Whitebox penetration testing with full source code access
- **Assessment Date:** December 15, 2025
- **Duration:** 11 minutes
- **Primary Endpoint:** /api/products (GET, POST, PUT methods)
- **Testing Focus:** API security, input validation, business logic, authentication, and authorization

3.2 Out of Scope

The following items were explicitly excluded from the scope of this assessment:

- Any domains, subdomains, or services not explicitly mentioned in the In Scope section
- Third-party services, libraries, or dependencies (unless directly exploitable through application code)
- Social engineering, phishing, or physical security testing
- Denial of Service (DoS) attacks or resource exhaustion testing
- Testing against production systems or live customer data
- Reverse engineering of compiled or obfuscated code
- Testing of infrastructure, network, or cloud provider security controls

4. Assets Tested

Endpoints Tested

Endpoint	HTTP Methods	Type	Authentication	Description
/api/products	GET, POST, PUT	API Endpoint	Not Required (GET), JWT Required (POST/PUT)	Handles product operations. GET requests retrieve all products with optional search filtering. POST requests create new products (admin only). PUT requests update existing products.

Applications Tested

Application Name	Type	Framework	Description
coffee-shop	Web API	Next.js / Node.js	RESTful API for managing coffee shop products, inventory, and orders. Built with Next.js API routes and PostgreSQL database backend.

5. Attack Surface Methodology

Pensar conducted a comprehensive attack surface analysis of the coffee-shop application to identify all exposed endpoints, authentication mechanisms, and potential attack vectors. The reconnaissance phase began with source code analysis, allowing Pensar to map the application architecture, identify API endpoints, and understand the authentication and authorization implementation.

Through static code analysis of the Next.js application structure, Pensar identified the primary API endpoint at /api/products, which handles product catalog operations. The endpoint supports multiple HTTP methods (GET, POST, PUT) with different authentication requirements. GET requests are publicly accessible, while POST and PUT requests require JWT authentication with admin privileges. Pensar documented the endpoint's request/response structure, including required fields, data types, and expected behavior.

Pensar analyzed the authentication mechanism and discovered that the application implements JWT-based authentication stored in HTTP-only cookies. The JWT tokens are verified using cryptographic signature validation, which Pensar confirmed through testing. The application enforces role-based access control, requiring admin privileges for product creation and modification operations. This authentication approach demonstrates security awareness, as it protects against common CSRF attacks through the use of JWT tokens rather than simple session cookies.

During the reconnaissance phase, Pensar examined the database schema and API request/response patterns to understand the data model and business logic. The product entity includes fields for name, description, price, stock_quantity, and server-generated timestamps. Pensar identified that the API accepts user-provided values for all product fields, including price and stock_quantity, which are critical business-logic fields that should be subject to strict validation.

The attack surface mapping revealed a single primary endpoint with limited external attack surface, but significant internal business logic complexity. The application's reliance on server-side validation for business-critical fields was identified as a key area requiring detailed testing. Pensar proceeded with focused testing on input validation, business logic enforcement, and authorization controls to identify specific vulnerabilities.

6. Pentesting Methodology

Pensar employed a systematic whitebox penetration testing approach, leveraging full access to the application source code and running environment to conduct comprehensive security testing. The testing methodology was structured around key security domains: authentication and authorization, input validation, business logic, and data integrity.

The initial phase focused on authentication and authorization testing. Pensar verified that JWT signature validation is properly implemented and that modified or forged tokens are correctly rejected by the API. Testing confirmed that the application enforces role-based access control, requiring admin privileges for product creation and modification. Pensar also tested for common JWT vulnerabilities, including the "none" algorithm attack and signature bypass techniques, confirming that these attacks are properly mitigated.

The second phase concentrated on input validation testing for the product creation endpoint. Pensar systematically tested the POST /api/products endpoint with various invalid inputs to identify validation gaps. Testing included attempts to create products with negative prices, negative stock quantities, invalid data types, and boundary value testing. Pensar discovered that the application fails to validate that the price field must be non-negative, allowing creation of products with negative prices. Similarly, the stock_quantity field lacks validation to ensure non-negative values.

Pensar tested for mass assignment vulnerabilities by attempting to inject additional fields (id, created_at) into product creation requests. The application correctly ignored these injected fields and used server-generated values, demonstrating proper handling of mass assignment attacks. Testing also included attempts to trigger SQL injection through various input vectors, which were properly mitigated through the use of parameterized queries.

The testing methodology included analysis of error handling and information disclosure. Pensar observed that invalid input (such as non-numeric values for numeric fields) triggers database errors that are returned to the client. While these errors do not expose sensitive information, they could be improved to provide more generic error messages. Testing also verified that the application properly handles edge cases and boundary conditions.

Throughout the assessment, Pensar documented specific exploitation techniques, proof-of-concept payloads, and the business impact of each identified vulnerability. The testing approach balanced breadth (covering multiple attack vectors) with depth (thoroughly validating each finding), ensuring comprehensive coverage of the application's security posture.

7. Findings

Finding #1: Missing Price Validation Allows Negative Product Prices

Risk Rating: Critical

CWE-20: Improper Input Validation | Endpoint: POST /api/products | Status: Open

Observation

Pensar identified a critical security issue where the /api/products endpoint (POST method) fails to implement server-side validation for product prices. An authenticated admin user can create products with negative prices by sending a POST request with a negative value in the "price" field. The server accepts and persists this invalid data without any validation or rejection, resulting in a critical business logic vulnerability that directly impacts financial integrity and inventory management.

Technical Explanation

This vulnerability exists because the application fails to implement server-side validation for the price field before persisting product data to the database. The root cause stems from insufficient input validation in the product creation handler. The application accepts the price value directly from the client request without verifying that it represents a valid, non-negative monetary amount. Due to the absence of business logic validation, any authenticated admin user can bypass the intended constraint that product prices must be positive values.

Exploitation

The vulnerability can be exploited through the following steps:

1. Obtain valid JWT authentication credentials with admin privileges
2. Craft a POST request to /api/products with a negative price value:

```
POST /api/products HTTP/1.1
Host: localhost:3000
Content-Type: application/json
Authorization: Bearer <valid-jwt-token>

{
  "name": "Exploit Product",
  "description": "Test product with negative price",
  "price": -50.00,
  "stock_quantity": 10
}
```

3. Send the request to the API endpoint
4. Observe that the server responds with HTTP 201 (Created) and persists the product with the negative price to the database

5. Verify the product was created by querying GET /api/products and confirming the negative price is stored

Impact Achieved

Through this vulnerability, Pensar was able to create products with negative prices in the database, demonstrating complete bypass of business logic validation. The server accepted and persisted invalid data without any error or warning, confirming the vulnerability is fully exploitable by authenticated administrators.

Impact Assessment

This critical vulnerability has severe business and technical implications. An attacker with admin privileges could create products with negative prices, which would result in the system issuing credits or refunds instead of charging customers for purchases. This could be exploited to facilitate fraud, money laundering, or unauthorized financial transfers.

The technical impact extends beyond simple data corruption. Systems that rely on product pricing data for financial reporting, inventory valuation, or revenue calculations would produce incorrect results. Negative prices could propagate through order processing systems, accounting systems, and financial statements, creating widespread data integrity issues.

The business impact is severe: financial fraud, revenue loss, inventory mismanagement, and potential regulatory compliance violations. If this application processes real financial transactions, the vulnerability could result in direct financial loss and legal liability.

Recommendation

Pensar recommends implementing comprehensive server-side validation for the price field:

1. **Implement Price Validation:** Add validation logic to ensure the price field is a positive number greater than zero. Reject any request with a price value less than or equal to zero with a clear error message.
2. **Use Validation Framework:** Implement validation using a schema validation library (e.g., Zod, Joi, or similar) to enforce type safety and business logic constraints at the API handler level.
3. **Database Constraints:** Add database-level constraints (CHECK constraint) to prevent negative prices from being stored, providing defense-in-depth protection.
4. **Code Example:**

```
// Recommended validation implementation
const createProductSchema = z.object({
  name: z.string().min(1),
  description: z.string(),
  price: z.number().positive("Price must be greater than zero"),
  stock_quantity: z.number().int().nonnegative()
});

// In the POST handler:
const validatedData = createProductSchema.parse(req.body);
// Proceed with database insertion only if validation passes
```

5. **Add Database Constraint:** Execute the following SQL to add a CHECK constraint:

```
ALTER TABLE products ADD CONSTRAINT price_positive CHECK (price > 0);
```

6. **Test Coverage:** Add unit tests and integration tests to verify that negative prices are rejected and that valid prices are accepted.
7. **References:** CWE-20: Improper Input Validation, OWASP Business Logic Vulnerability

Finding #2: Missing Validation for Negative Stock Quantities in Product Creation

Risk Rating: High

CWE-20: Improper Input Validation | Endpoint: POST /api/products | Status: Open

Observation

Pensar discovered a significant security concern where the POST /api/products endpoint fails to validate that stock_quantity is non-negative before persisting to the database. An authenticated admin user can create products with negative stock quantities by sending a POST request with a negative value in the "stock_quantity" field. The server accepts and persists this invalid data without any validation or rejection, resulting in a critical business logic vulnerability affecting inventory management systems.

Technical Explanation

This vulnerability exists because the application fails to implement server-side validation for the stock_quantity field before persisting product data to the database. The root cause stems from insufficient input validation in the product creation handler, similar to the price validation issue. The application accepts the stock_quantity value directly from the client request without verifying that it represents a valid, non-negative integer. Due to the absence of business logic validation, any authenticated admin user can create products with negative stock quantities, which violates fundamental inventory management principles.

Exploitation

The vulnerability can be exploited through the following steps:

1. Obtain valid JWT authentication credentials with admin privileges
2. Craft a POST request to /api/products with a negative stock_quantity value:

```
POST /api/products HTTP/1.1
Host: localhost:3000
Content-Type: application/json
Authorization: Bearer <valid-jwt-token>

{
  "name": "Exploit Product",
  "description": "Test product with negative stock",
  "price": 25.00,
  "stock_quantity": -100
}
```

3. Send the request to the API endpoint
4. Observe that the server responds with HTTP 201 (Created) and persists the product with the negative stock quantity to the database

5. Verify the product was created by querying GET /api/products and confirming the negative stock quantity is stored

Impact Achieved

Through this vulnerability, Pensar was able to create products with negative stock quantities in the database, demonstrating complete bypass of inventory management validation. The server accepted and persisted invalid data without any error or warning, confirming the vulnerability is fully exploitable by authenticated administrators.

Impact Assessment

This high-severity vulnerability disrupts inventory management and business operations. Negative stock quantities represent an impossible state in real-world inventory systems and indicate a fundamental data integrity issue. An attacker with admin privileges could create products with negative stock, which would confuse inventory tracking systems and potentially allow overselling of products.

The technical impact includes data integrity violations, incorrect inventory calculations, and potential system errors when processing orders against negative stock. Systems that calculate available inventory, generate purchase orders, or manage warehouse operations would produce incorrect results based on corrupted stock data.

The business impact includes inventory mismanagement, potential overselling of products, customer dissatisfaction, and operational disruption. If the application manages real inventory, negative stock quantities could lead to fulfillment failures and customer service issues.

Recommendation

Pensar recommends implementing comprehensive server-side validation for the stock_quantity field:

1. **Implement Stock Quantity Validation:** Add validation logic to ensure the stock_quantity field is a non-negative integer. Reject any request with a stock_quantity value less than zero with a clear error message.
2. **Use Validation Framework:** Implement validation using a schema validation library (e.g., Zod, Joi, or similar) to enforce type safety and business logic constraints at the API handler level.
3. **Database Constraints:** Add database-level constraints (CHECK constraint) to prevent negative stock quantities from being stored, providing defense-in-depth protection.
4. **Code Example:**

```
// Recommended validation implementation
const createProductSchema = z.object({
  name: z.string().min(1),
  description: z.string(),
  price: z.number().positive(),
  stock_quantity: z.number().int().nonnegative("Stock quantity cannot be negative")
});

// In the POST handler:
const validatedData = createProductSchema.parse(req.body);
// Proceed with database insertion only if validation passes
```

5. **Add Database Constraint:** Execute the following SQL to add a CHECK constraint:

```
ALTER TABLE products ADD CONSTRAINT stock_quantity_nonnegative CHECK (stock_quantity >= 0);
```

6. **Implement Stock Adjustment Logic:** For stock updates, implement separate endpoints for stock adjustments (add/remove stock) rather than allowing direct modification of the stock_quantity field.
7. **Test Coverage:** Add unit tests and integration tests to verify that negative stock quantities are rejected and that valid quantities are accepted.
8. **References:** CWE-20: Improper Input Validation, OWASP Business Logic Vulnerability

8. Additional Observations

Positive Security Observations

JWT Authentication Implementation

Pensar observed that the application implements JWT-based authentication with proper cryptographic signature verification. The application correctly rejects tokens with invalid signatures and does not accept the "none" algorithm, which demonstrates security awareness. The use of JWT tokens stored in HTTP-only cookies provides protection against common CSRF attacks and XSS-based token theft.

Role-Based Access Control

The application implements role-based access control (RBAC) to restrict product creation and modification to admin users. This demonstrates proper authorization enforcement and prevents unauthorized users from modifying product data. The RBAC implementation should be maintained and extended to other sensitive operations.

Mass Assignment Protection

Testing revealed that the application properly handles mass assignment attacks by ignoring injected fields (such as id and created_at) and using server-generated values instead. This demonstrates good security practice in preventing attackers from manipulating system-generated fields.

Recommendations for Defense-in-Depth

Error Message Improvement

While not a critical vulnerability, the application could improve error handling by providing more generic error messages to clients. Currently, database errors (such as PostgreSQL error codes) may be exposed to clients, which could leak information about the database structure. Pensar recommends implementing a centralized error handling middleware that translates database errors into generic, user-friendly error messages.

Input Validation Framework

Pensar recommends implementing a comprehensive input validation framework across all API endpoints. The current implementation relies on ad-hoc validation, which has led to the identified vulnerabilities. A schema-based validation approach (using libraries like Zod or Joi) would provide consistent, maintainable validation across the application.

API Rate Limiting

The application should implement rate limiting on API endpoints to prevent abuse and brute-force attacks. While not currently exploitable, rate limiting provides defense-in-depth protection against various attack scenarios.

9. Conclusion

The coffee-shop application contains critical security vulnerabilities that require immediate remediation. The identified vulnerabilities stem from insufficient input validation on business-critical fields, specifically the price and stock_quantity fields in the product creation endpoint. These vulnerabilities allow authenticated administrators to create products with invalid data, compromising data integrity and enabling potential fraud.

The presence of a critical vulnerability (negative price validation) represents an urgent security concern that must be addressed before the application handles real financial transactions. The vulnerability could be exploited to manipulate financial records, facilitate fraud, and cause direct financial loss. The high-severity vulnerability (negative stock quantity validation) represents a significant operational risk that could disrupt inventory management and customer fulfillment.

While the assessment identified only two vulnerabilities, both stem from a common root cause: insufficient input validation. Pensar recommends implementing a comprehensive input validation framework across all API endpoints to prevent similar vulnerabilities from being introduced in the future. The application demonstrates some positive security practices, including JWT authentication, role-based access control, and mass assignment protection, which should be maintained and extended.

Recommended Remediation Timelines

Pensar recommends prioritizing remediation based on risk severity, consistent with industry standards from NIST, CISA, and leading cybersecurity frameworks:

Severity Level	CVSS Range	Remediation Target	Justification
Critical	9.0-10.0	15 days	These vulnerabilities pose immediate risk and require emergency patching. The negative price validation vulnerability directly impacts financial integrity and must be remediated urgently.
High	7.0-8.9	30 days	Significant security concerns requiring prompt attention to prevent exploitation. The negative stock quantity vulnerability disrupts inventory management and should be addressed within one month.
Medium	4.0-6.9	90 days	Should be addressed in regular maintenance cycles and quarterly security updates.
Low	0.1-3.9	120 days	Defense-in-depth measures to strengthen overall security posture, addressed in regular maintenance.

Timeline recommendations are based on CISA Binding Operational Directive 19-02, NIST Special Publication 800-53 Control SI-2, and GSA vulnerability management guidance.

Next Steps

Pensar recommends the following immediate actions:

1. **Immediate (Days 1-3):** Implement input validation for the price field to reject negative values. This is the highest priority due to the critical severity and financial impact.
2. **Short-term (Days 4-15):** Implement input validation for the stock_quantity field and add database-level constraints for both fields.
3. **Medium-term (Days 16-30):** Implement a comprehensive input validation framework across all API endpoints to prevent similar vulnerabilities.
4. **Ongoing:** Establish a security testing process to validate fixes and prevent regression of identified vulnerabilities.

Final Assessment

The coffee-shop application requires urgent security remediation before deployment to production or handling of real financial transactions. The identified vulnerabilities represent critical business risks that could result in financial fraud, data corruption, and operational disruption. However, the application demonstrates foundational security practices that provide a solid foundation for improvement. With focused remediation efforts on input validation and implementation of the recommended security controls, the application's security posture can be significantly improved.

Pensar is available to conduct a retest assessment following remediation to verify that the identified vulnerabilities have been successfully addressed and to validate the effectiveness of the implemented security controls.

Josh Kotrous

Pentester Signature

Josh Kotrous

Pentester Printed Name

December 15, 2025

Date signed