WEB APPLICATION SECURITY ASSESSMENT REPORT

TARGET APPLICATION:
OWASP JUICE SHOP *(INTENTIONALLY VULNERABLE APPLICATION)*

PROGRAM:
FUTURE INTERNS CYBERSECURITY INTERNSHIP

TASK 1:
WEB APPLICATION SECURITY TESTING

PREPARED BY:
ZANNU OPEYEMI EMMANUEL

DATE: DECEMBER 2025

TASK SUMMARY:

This assessment evaluated the security posture of the OWASP Juice Shop application through a comprehensive black-box penetration test. Utilizing Burp Suite for traffic interception and manual validation, the engagement identified significant security flaws, specifically User Enumeration and Information Leakage. These vulnerabilities present a tangible risk, effectively widening the attack surface for credential harvesting and account takeover. All findings have been risk-rated and mapped against the OWASP Top 10 (2021) framework to facilitate prioritized remediation.

TOOLS UTILIZED:

- **Kali Linux**
- **OWASP Juice Shop**
- **Burp Suite**

PROCEDURE:

The security assessment was conducted using Kali Linux, OWASP Juice Shop, and Burp Suite as the primary tools. The OWASP Juice Shop application was deployed locally within a Linux-based virtual machine, with all necessary dependencies including Node.js and npm installed via the Linux command-line interface. This setup provided a controlled yet realistic environment for simulating real-world web application attack scenarios.

The locally hosted instance of the application was designated as the target system for both automated vulnerability scanning and manual security testing. Burp Suite was employed as the central testing platform, where its active scanning capabilities were used to systematically identify application endpoints, input fields, and user interaction flows, enabling the discovery of potential security weaknesses across the application's attack surface.

All vulnerabilities identified during the assessment were thoroughly analyzed and mapped to their respective categories within the OWASP Top 10 framework. Based on this classification, appropriate remediation and mitigation recommendations were proposed to address the identified risks and improve the overall security posture of the application.

- **VULN-01: Feedback Replay & Identity Impersonation**

Severity: Medium

OWASP Category: A01:2021 – Broken Access Control

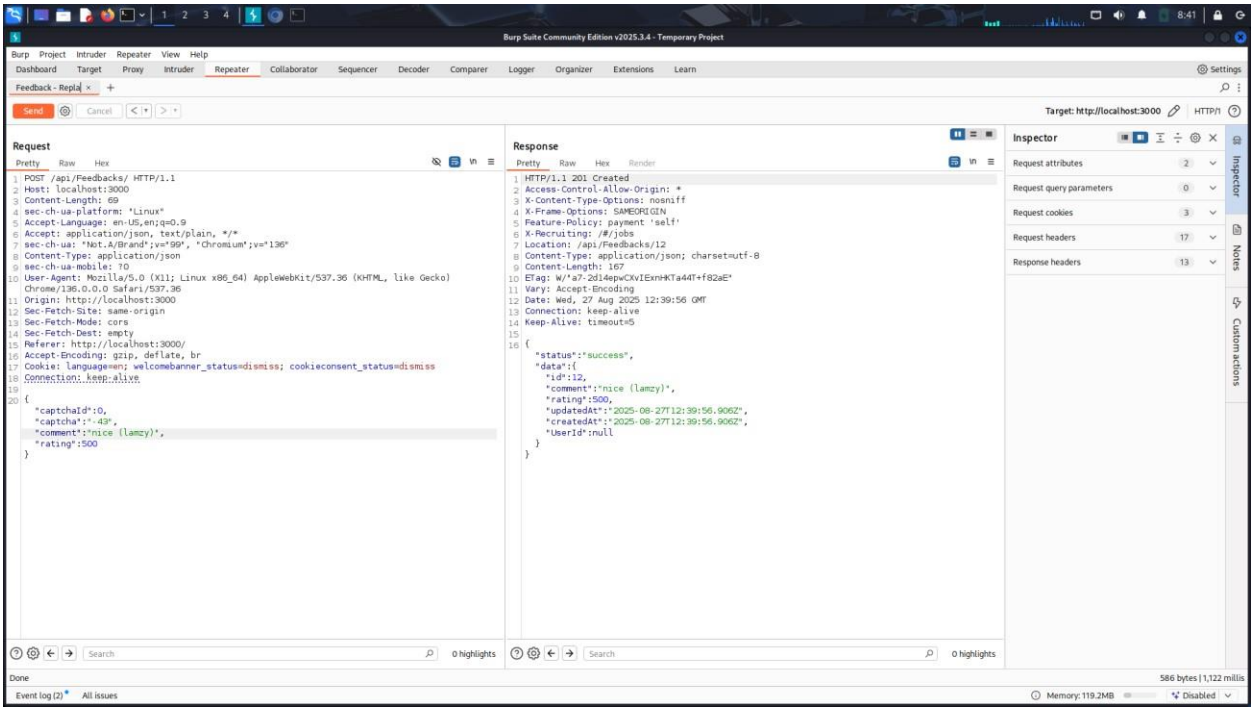
Description: The application's feedback mechanism lacks proper idempotency controls and robust session validation.

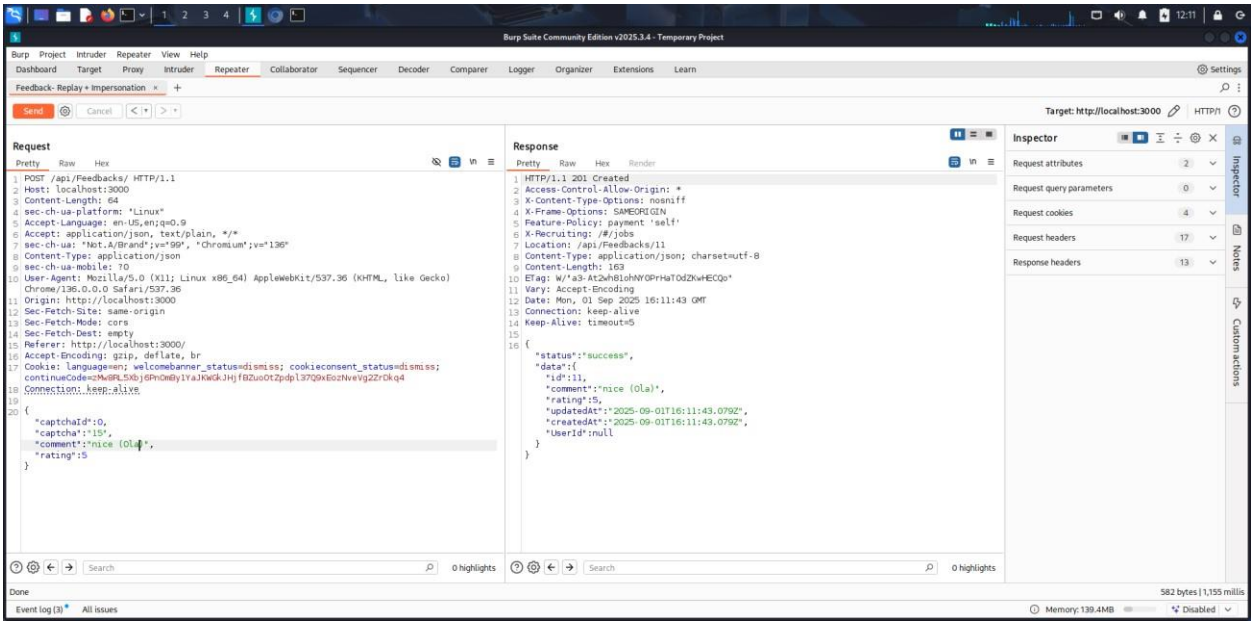
- **Replay Attack: The application accepts identical HTTP requests multiple times without validation (e.g., missing unique nonces or timestamps). This allows an attacker to capture a legitimate request and re-send it repeatedly.**
- **Impersonation: The user identity associated with the feedback is derived from client-side parameters rather than the secure server-side session token. An attacker can manipulate the UserId or Author fields in the request body to submit feedback on behalf of another user.**

Impact:

- **Data Integrity Degradation: The database can be flooded with duplicate or spam submissions, skewing analytics and consuming storage resources.**
- **Reputational Damage: Attackers can submit offensive or malicious feedback attributable to innocent users, potentially leading to incorrect account bans or loss of user trust.**

Evidence:





Remediation & Mitigation Strategies:

- **Enforce Server-Side Identity Validation: Do not rely on user-submitted parameters (e.g., UserId in the JSON body) to identify the author. Instead, strictly derive the user's identity from the authenticated session token (e.g., JWT or Session Cookie) on the server side.**
- **Implement Anti-Replay Controls: Integrate Idempotency Keys or unique, one-time-use tokens (nonces) for form submissions. Additionally, verify the timestamp of requests to ensure they fall within an acceptable time window, rejecting any stale or duplicate requests.**
- **Apply Rate Limiting: Implement strict rate limiting on the feedback endpoint (e.g., "max 3 submissions per minute per user") to prevent automated spamming and resource exhaustion.**

➢ **VULN-02: SQL Injection (Authentication Bypass)**

Severity: High (Critical)

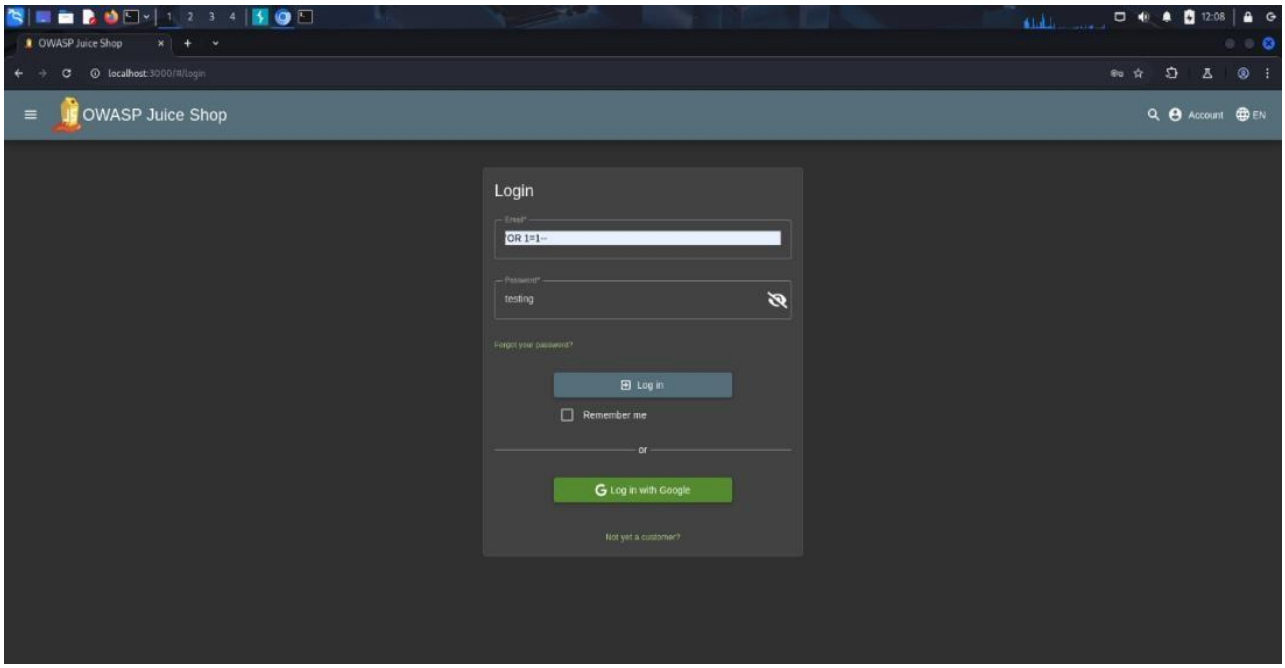
OWASP Category: A03:2021 – Injection

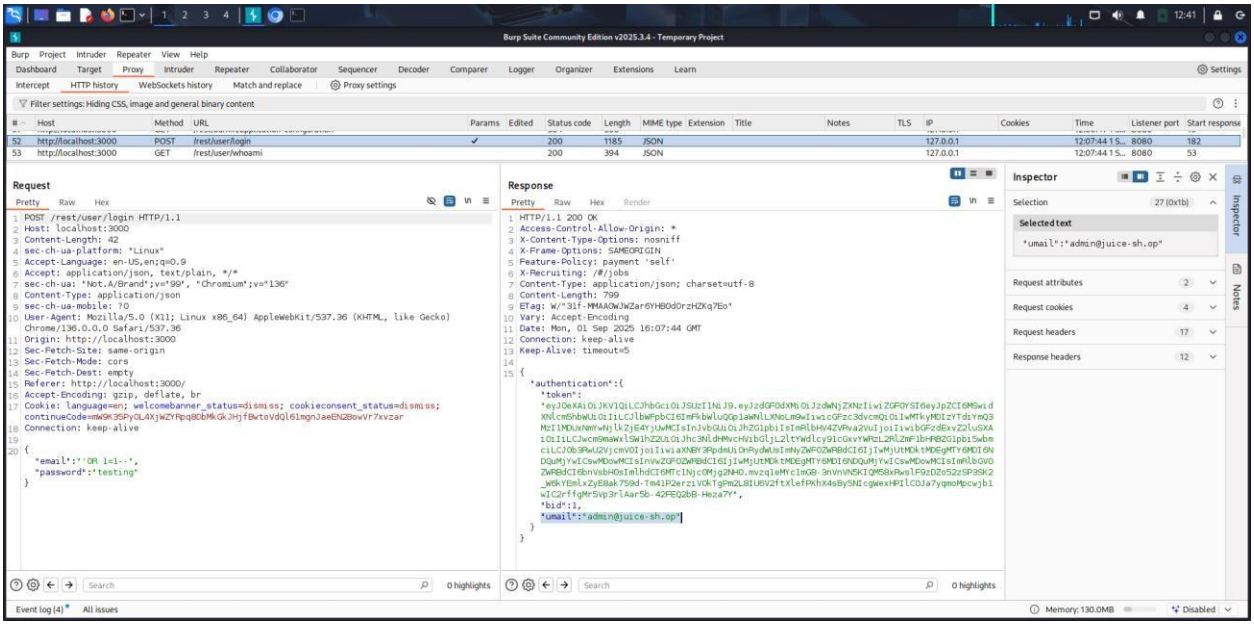
Description: The application's login mechanism fails to properly sanitize user-supplied input before processing it in the backend database. Specifically, the email parameter is vulnerable to SQL Injection (SQLi). By injecting a standard tautology payload (e.g., ' OR 1=1 --), an attacker can manipulate the structure of the SQL query. The injected code alters the query logic to evaluate as "True" regardless of the password provided, effectively bypassing the authentication check.

Impact:

- **Authentication Bypass: Unauthenticated attackers can log in as any user, including the Administrator, without valid credentials.**
- **Total System Compromise: Gaining administrative access allows the attacker to view all user data, modify application settings, and potentially execute further attacks against the server.**

Evidence:





Remediation & Mitigation Strategies:

- **Primary Defense: Parameterized Queries (Prepared Statements) The most effective defense against SQL injection is the use of Parameterized Queries (also known as Prepared Statements). This technique ensures that the database treats user input strictly as data, never as executable code, effectively neutralizing the attack regardless of the input content.**
- **Defense in Depth: Input Validation & Sanitization Implement strict Allow-listing (white-listing) for all user inputs. Ensure that input adheres to expected formats (e.g., email addresses must contain an @ symbol and a domain). Any input containing unexpected characters or SQL keywords should be rejected or sanitized before processing.**
- **Principle of Least Privilege (Database Hardening) Configure the database application user with the minimum necessary privileges required to function. For example, the account used by the web application should strictly have SELECT, INSERT, and UPDATE permissions on specific tables, and should never have administrative rights (like DROP TABLE or GRANT).**

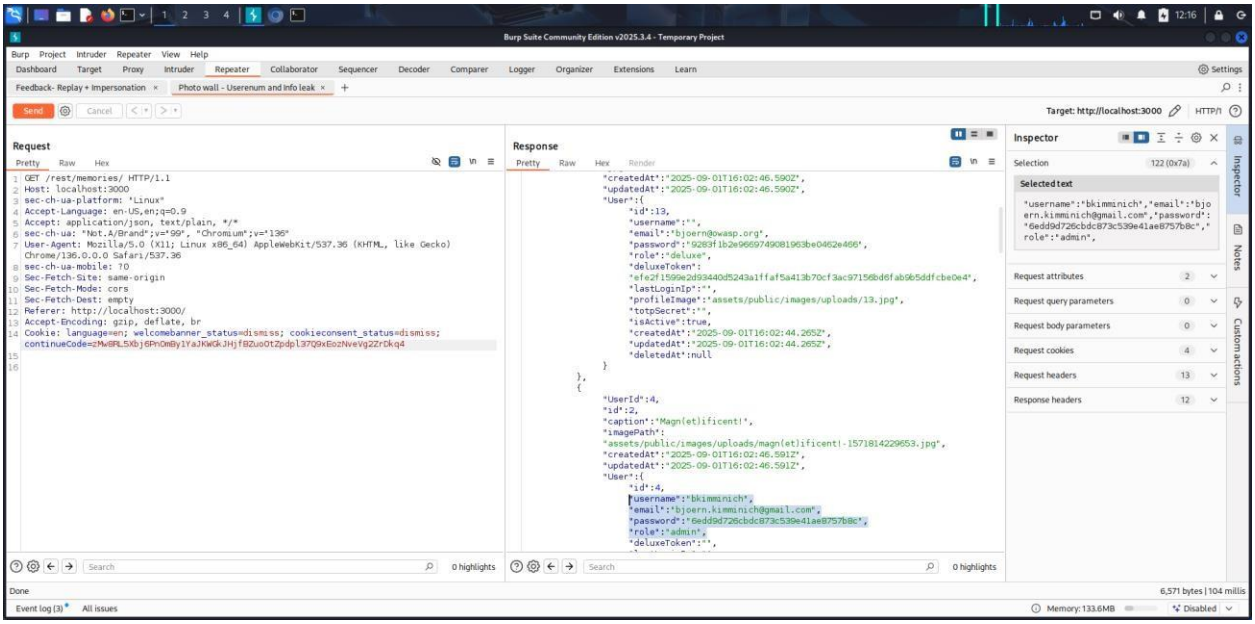
➢ **VULN-03: User Enumeration & Information Leakage**

Severity: Medium

OWASP Category: A05:2021 – Security Misconfiguration / A07:2021 – Identification and Authentication Failures

Description: The application's "Photo Wall" functionality exhibits inconsistent response behavior when querying user data.

- **User Enumeration: By observing subtle differences in server responses (such as error messages, response times, or HTTP status codes), an attacker can distinguish between valid and invalid user accounts. This allows for the harvesting of a verified list of usernames or email addresses.**



Remediation & Mitigation Strategies:

- **Standardize Authentication Responses: Configure the application to return identical, generic error messages (e.g., "Invalid username or password") regardless of whether the account exists or not. Ensure that the response time for valid and invalid attempts is consistent to prevent timing-based enumeration attacks.**
- **Implement Rate Limiting & Account Lockout: Enforce strict rate limiting on all authentication and query endpoints to detect and block automated enumeration attempts. Implement temporary account lockouts after a defined threshold of failed attempts to impede brute-force attacks.**
- **Secure Error Handling (Suppress Verbose Output): Disable verbose error reporting in the production environment. Ensure that stack traces, system version numbers, and debug information are never exposed in HTTP responses or API headers to prevent information leakage.**

- **VULN-04: Insecure Token Design (Weak JWT Implementation)**

Severity: High

OWASP Category: A02:2021 – Cryptographic Failures

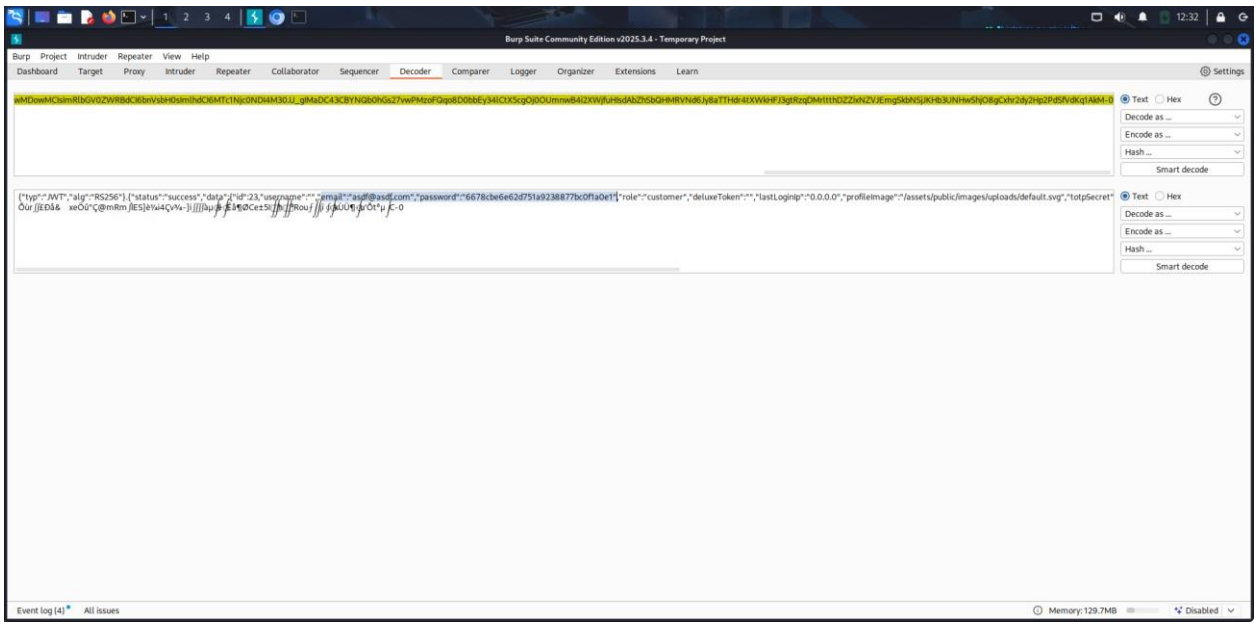
Description: The application utilizes JSON Web Tokens (JWT) for stateless session management. However, a security analysis of the token payload reveals a critical design flaw:

- **Sensitive Data Exposure: The token payload contains the user's email address and password hash. JWTs are merely Base64-encoded, not encrypted, meaning this data is readable by anyone who possesses the token.**
- **Weak Cryptography: The password embedded within the token is hashed using MD5. MD5 is a cryptographically broken algorithm that is considered unsafe for password storage due to its vulnerability to collision attacks and high-speed offline cracking.**

Impact:

- **Credential Compromise: An attacker who intercepts the token (e.g., via Man-in-the-Middle or XSS) can easily decode the payload and use rainbow tables or brute-force tools to reverse the MD5 hash, recovering the user's plain-text password.**
- **Account Takeover: With the recovered password, the attacker can gain unauthorized access to the user's account and potentially pivot to other services if password reuse is present.**

Evidence:



Remediation & Mitigation Strategies:

- **Token Hygiene (Data Minimization):**

Never store sensitive credentials (such as passwords, hash fragments, or PII) within the JWT payload. Since JWTs are typically only Base64-encoded (not encrypted), their contents are visible to anyone who intercepts them1. Limit claims to non-sensitive identifiers, such as the Subject ID (sub) or authorization scopes.

- **Upgrade Password Hashing Standards:**

Immediately deprecate the use of MD5 for password storage3. Implement robust, slow-hashing algorithms designed specifically to resist GPU-based brute-force and rainbow table attacks. The current industry standards are Argon2id, bcrypt, or PBKDF24.

- **Enforce Strong Cryptographic Signing:**

Configure the JWT issuance service to use strong signing algorithms, such as RS256 (Asymmetric RSA) or HS256 (HMAC with SHA-256) with a sufficiently complex secret key5. Ensure that the server validates the signature on every request to prevent token tampering.

- **VULN-05: INSECURE DIRECT OBJECT REFERENCE (IDOR)**

Severity: High

OWASP Category: A01:2021 – Broken Access Control

Description:

The application suffers from Insecure Direct Object References (IDOR) across multiple API endpoints.

The server exposes direct references to internal database objects (specifically basketId and userId) in the URL or API parameters. Crucially, the application fails to perform server-side access control checks to verify that the user requesting the resource is actually the owner of that resource.

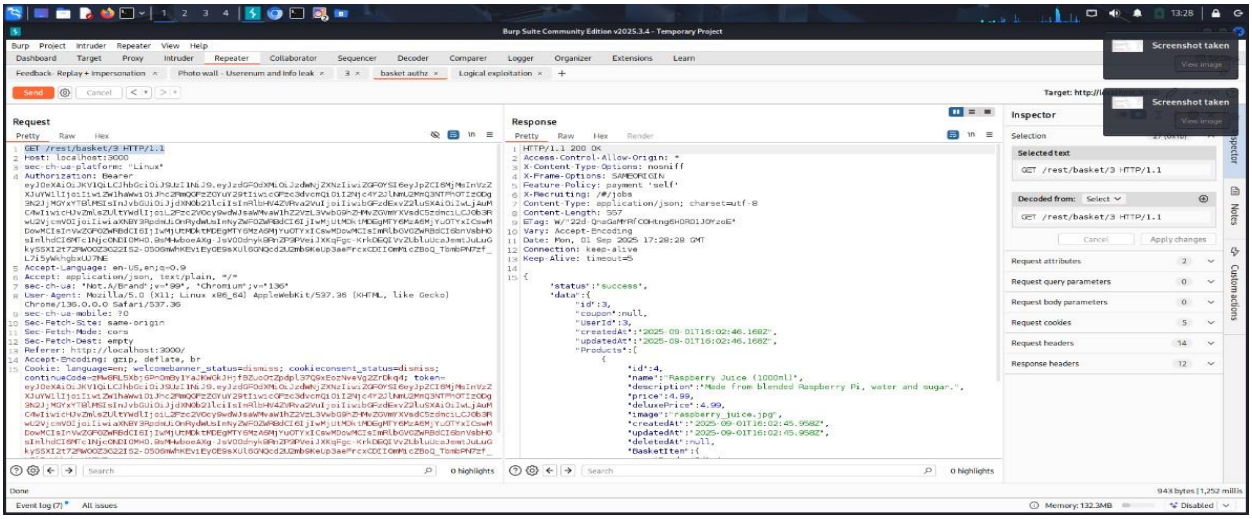
By manipulating these predictable parameters (e.g., changing basketId=1 to basketId=2), an attacker can bypass authorization mechanisms. This results in Horizontal Privilege Escalation, allowing them to access or modify data belonging to other users.

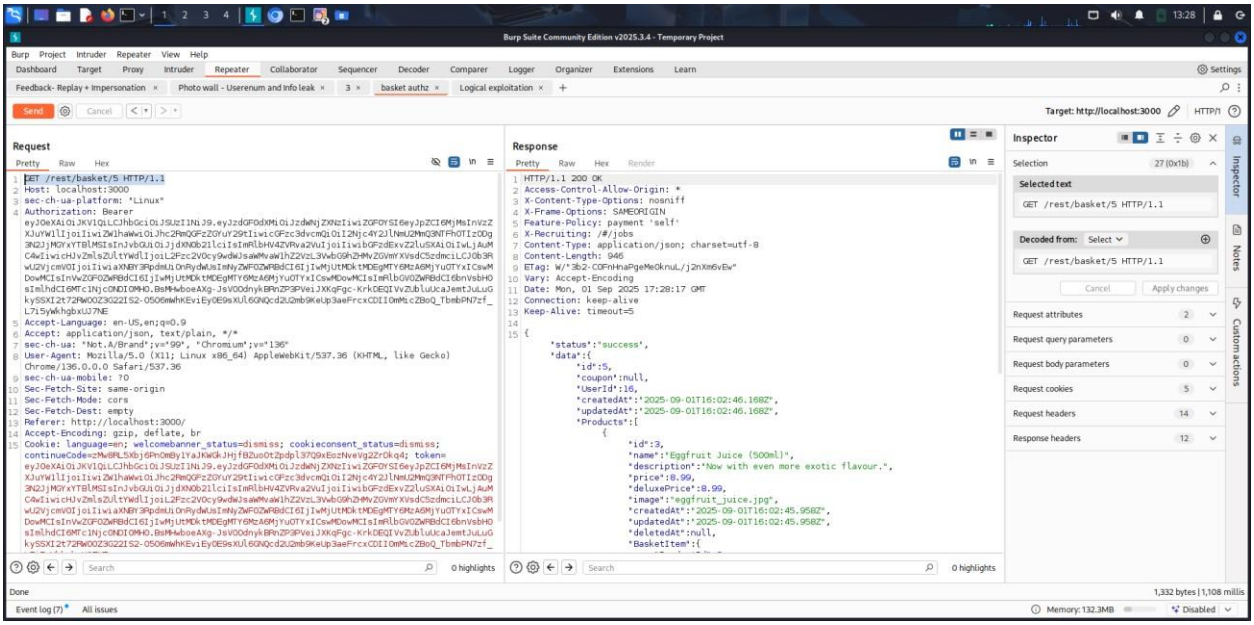
Impact:

- **Confidentiality Breach: Unauthorized access to sensitive user data, including shopping history, personal details (PII), and account settings.**

- **Data Manipulation: Attackers may be able to modify the contents of other users' shopping baskets or alter their account information.**

Evidence:





Remediation & Mitigation Strategies:

- **Enforce Server-Side Authorization Checks:**
  Implement mandatory access control checks at the code level for every request that accesses a data object. Before returning any data, the server must verify that the currently authenticated user has the specific permissions required to access the requested resource ID (e.g., if (request.userId != resource.ownerId) throw 403 Forbidden).
- **Use Indirect Object References (Reference Maps):**
  Avoid exposing raw, sequential database keys (like 1001, 1002) in URLs or API calls. Instead, use cryptographically random replacements, such as

GUIDs/UUIDs (e.g., a1b2-c3d4-e5f6), or implement a temporary mapping layer (Reference Map) that translates user-specific tokens to internal database IDs.

- **Implement "Ownership" Validation:**
  Ensure that logic specifically binds data to the user session. For example, when fetching a "Basket," the query should look like SELECT * FROM Baskets WHERE id = ? AND owner_id = current_session_user, rather than just querying by ID alone.

➢ **VULN-06: Business Logic Flaw (Payback Functionality)**

Severity: Medium

OWASP Category: A04:2021 – Insecure Design

Description:

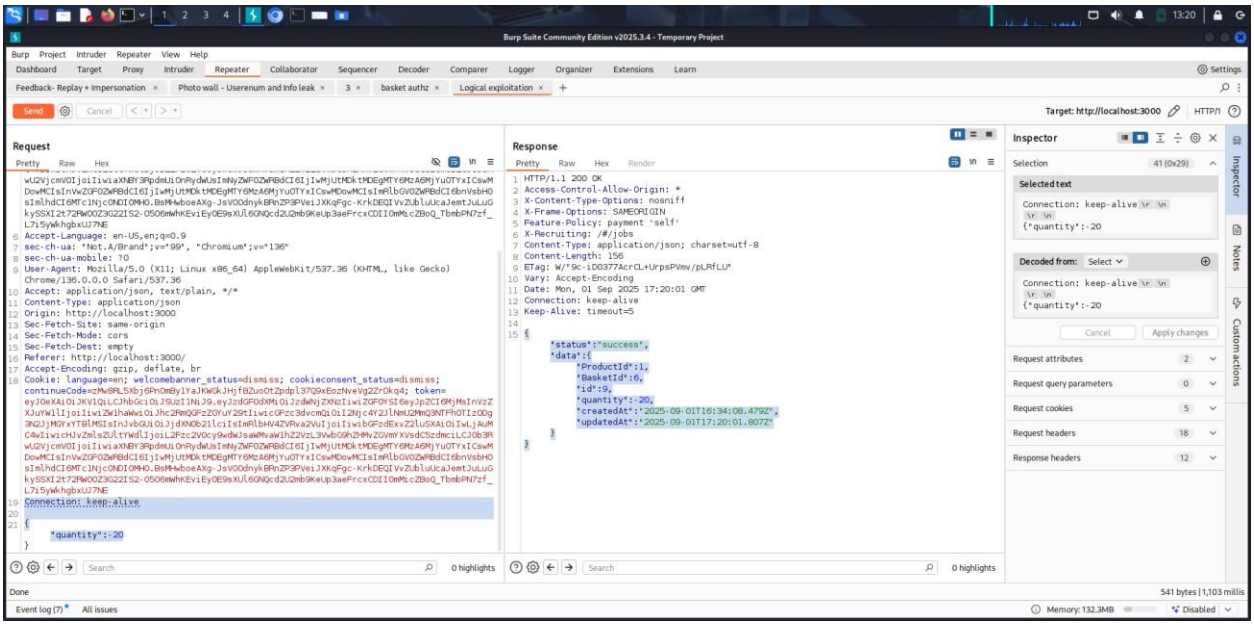
The application's payment reconciliation system contains a critical business logic flaw within the refund ("Payback") mechanism.

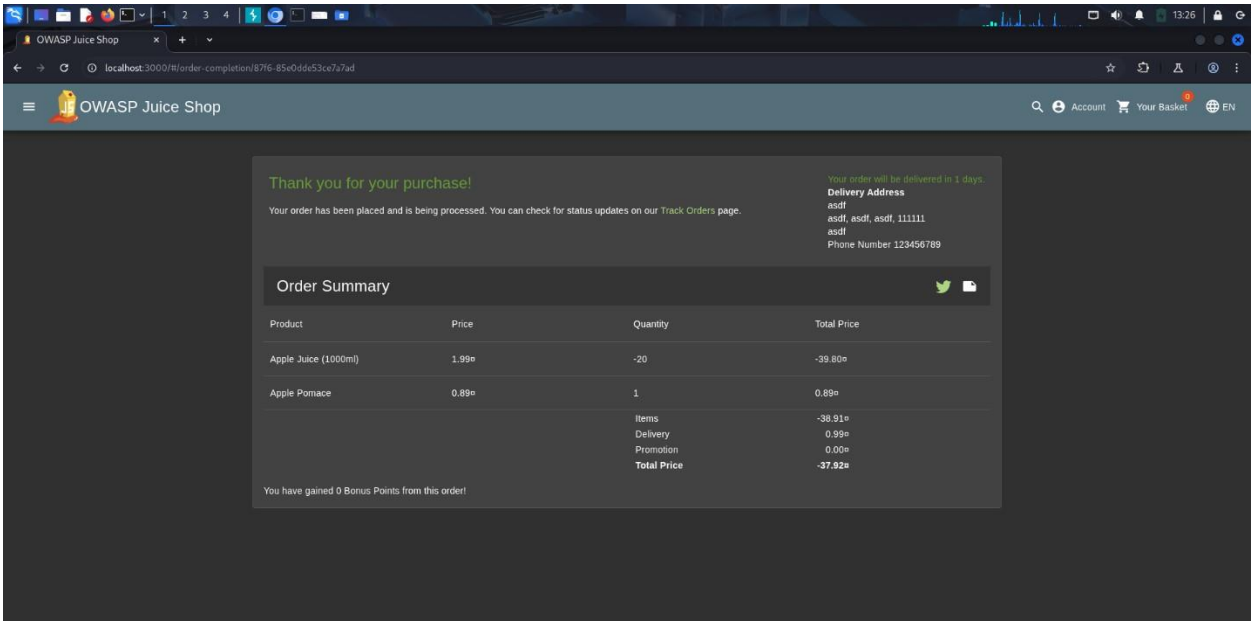
The application fails to enforce boundary checks or validate refund limits against the original transaction value. By manipulating input parameters during a return request—specifically by submitting negative quantities or values exceeding the original purchase—an attacker can reverse the arithmetic logic. This allows the user to credit their account with illegitimate funds rather than debiting it.

Impact:

- **Financial Fraud: Attackers can artificially inflate their wallet balance ("free funds"), leading to direct financial loss and inventory theft[3].**
- **Integrity Violation: The vulnerability undermines the reliability of the payment ledger, rendering transaction history and accounting inaccurate.**

Evidence:





Remediation & Mitigation Strategies:

- **Enforce Strict Transaction Logic Validation:**

Implement server-side checks to validate the integrity of every refund operation. Ensure that the refund amount never exceeds the original transaction value and that the quantity returned is positive and matches the purchased items.

- **Implement Boundary & Negative Value Checks:**

Sanitize all numerical inputs to reject negative values or integers that fall outside expected ranges (e.g., quantity > 0). Ensure arithmetic operations (like balance =

balance - refund) cannot overflow or underflow to create positive credit where a debit was intended.

- **Transactional Integrity (ACID Compliance):**

Wrap financial operations in atomic database transactions. Verify the final state of the user's balance and inventory before committing the transaction. If any logic check fails (e.g., resulting in an invalid negative balance), the entire transaction should roll back.

VULNERABILITY RISK MAPPING (OWASP TOP 10)

Vulnerability Name	OWASP Top 10 (2021) Category	Risk Severity
SQL Injection (Login Bypass)	A03:2021 – Injection	High
Insecure Token Design (Weak JWT)	A02:2021 – Cryptographic Failures	High
Insecure Direct Object Reference (IDOR)	A01:2021 – Broken Access Control	High
Replay & Impersonation (Feedback)	A01:2021 – Broken Access Control	Medium
User Enumeration & Info Leakage	A05:2021 – Security Misconfiguration	Medium
Business Logic Flaw (Payback)	A04:2021 – Insecure Design	Medium

CONCLUSION

The security assessment of the OWASP Juice Shop application revealed six significant vulnerabilities, ranging from High to Medium severity. Critical issues such as SQL Injection and Cryptographic Failures (Weak JWT) pose an immediate threat to the confidentiality and integrity of user data. Additionally, logic flaws in the payment system and broken access controls expose the application to financial fraud and unauthorized data access.

These findings align directly with the OWASP Top 10 (2021) risk categories, highlighting systemic weaknesses in input validation, access control enforcement, and secure design principles. To improve the application's security posture, it is recommended that the development team prioritize the implementation of parameterized queries, strong cryptographic standards, and robust server-side validation logic.