Performing a Security Audit of REST APIs and Testing for Vulnerabilities

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

DARBHA SUDHA VAISHNAVI – AP22110010201

Under the Guidance of

Dr. Bhaskara Santosh Egala

SRM University–AP
Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240
April 2025

Table of Contents

1. Introduction

In today’s interconnected world, applications heavily rely on REST APIs (Representational State Transfer Application Programming Interfaces) to facilitate communication and exchange data seamlessly across different platforms and services. APIs enable diverse functionalities, from fetching data to executing business logic, forming the backbone of modern web applications and mobile apps. However, the convenience and flexibility that APIs offer also introduce significant security risks. Despite their importance, many APIs are either left unprotected or have weak security measures, making them prime targets for cyberattacks.

These vulnerabilities can lead to severe consequences, such as unauthorized access to sensitive information, data breaches, service disruptions, or even full system compromise. Consequently, securing REST APIs is essential for maintaining user trust, preserving data integrity, and safeguarding application functionality. Attackers continuously exploit weaknesses in API security to launch malicious activities like SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and more.

This project aims to assess the security posture of REST APIs by conducting a comprehensive security audit and testing for various vulnerabilities. For this purpose, the OWASP Juice Shop platform has been utilized. OWASP Juice Shop is a deliberately insecure web application designed for training and testing web application security practices. By exploiting vulnerabilities in the Juice Shop's REST APIs, this project will simulate potential real-world attacks, identify critical flaws, and provide mitigation recommendations. Ultimately, the goal is to raise awareness about the importance of API security and guide developers toward more secure API design and implementation.

2. Problem Statement

As modern web applications increasingly rely on REST APIs for communication between the client and server, the security of these APIs becomes critical. However, many applications expose sensitive functionality without adequate protections, making them vulnerable to various attacks such as SQL Injection, Cross-Site Scripting (XSS), Insecure Direct Object Reference (IDOR), Cross-Site Request Forgery (CSRF), and Brute Force Attacks. The lack of proper input validation, authentication, and authorization checks puts user data and system integrity at serious risk. This project focuses on identifying such security flaws in a controlled environment to raise awareness and recommend mitigations.

3. Tools and Technologies Used

List the main tools and briefly describe their role:

1. **OWASP Juice Shop** – A deliberately insecure web app used to simulate real-world vulnerabilities.
2. **Burp Suite** – A web vulnerability scanner and proxy tool used to intercept, manipulate, and test HTTP requests and responses.

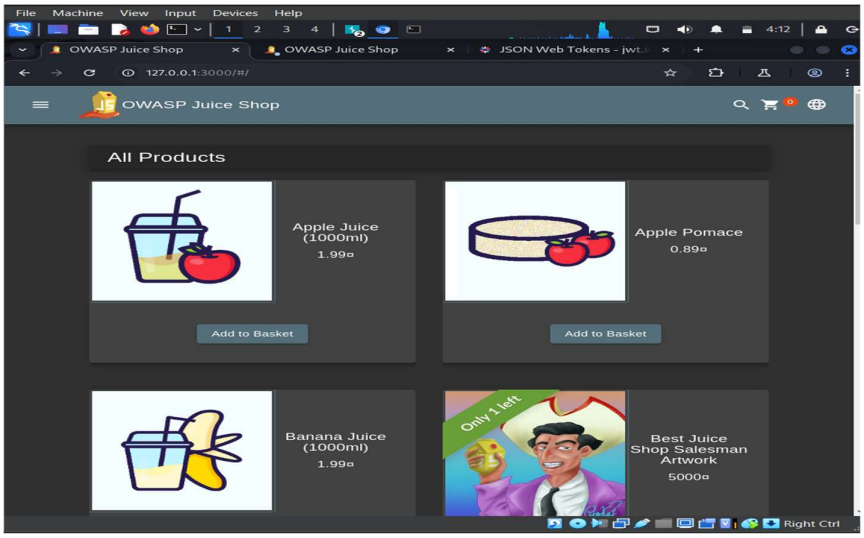
4. Methodology

A step-by-step outline of how the audit was conducted:

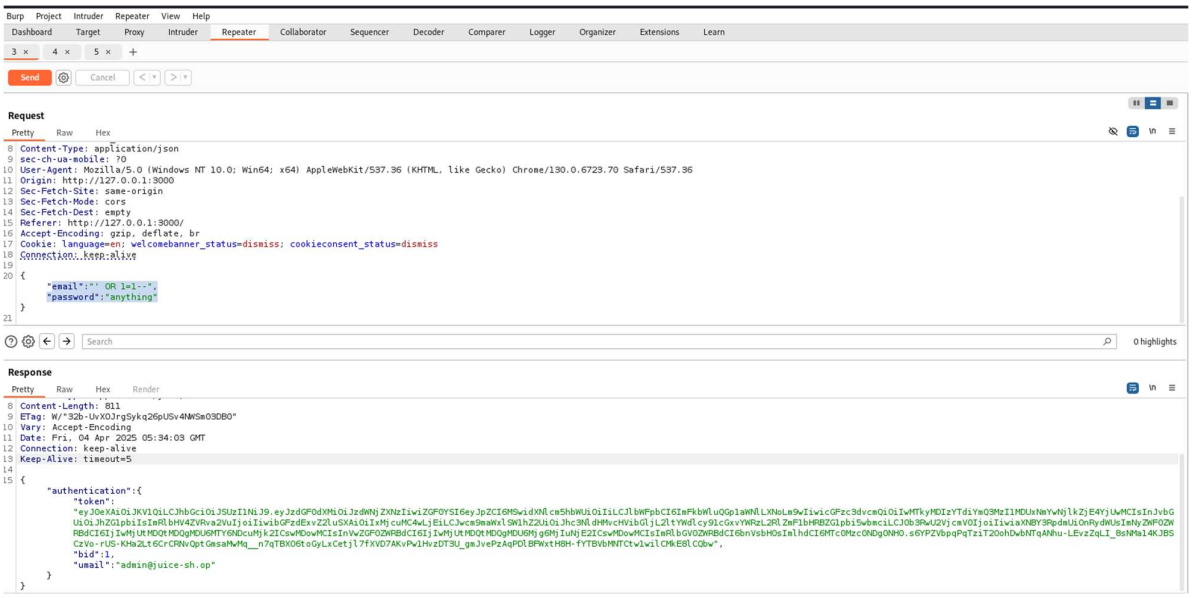
1. Setup OWASP Juice Shop locally using Kali Linux and VirtualBox.
2. Configured Burp Suite to intercept web traffic.
3. Identified target API endpoints through HTTP History.
4. Used Repeater and Intruder to test various input cases for vulnerabilities.
5. Recorded and documented findings for each tested vulnerability.

5. How the Solution is Achieved

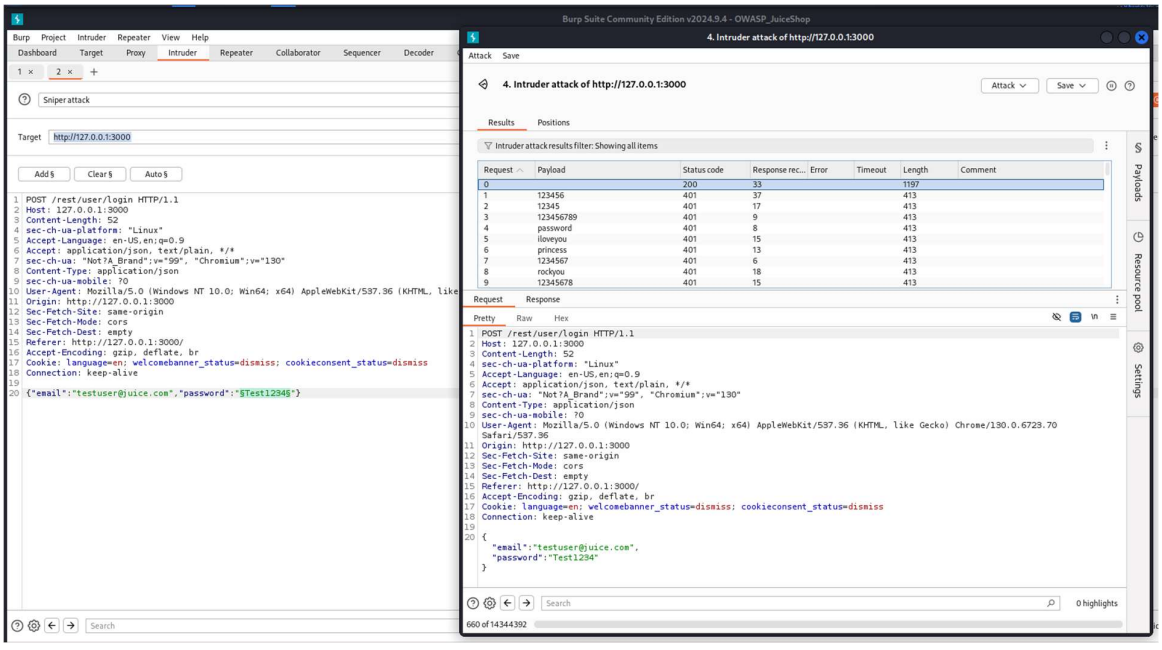
To perform the security audit, the project used **OWASP Juice Shop**, a vulnerable e-commerce web app, and **Burp Suite**, a powerful web security testing tool. Specific API endpoints were identified and tested for vulnerabilities by intercepting and manipulating API requests.



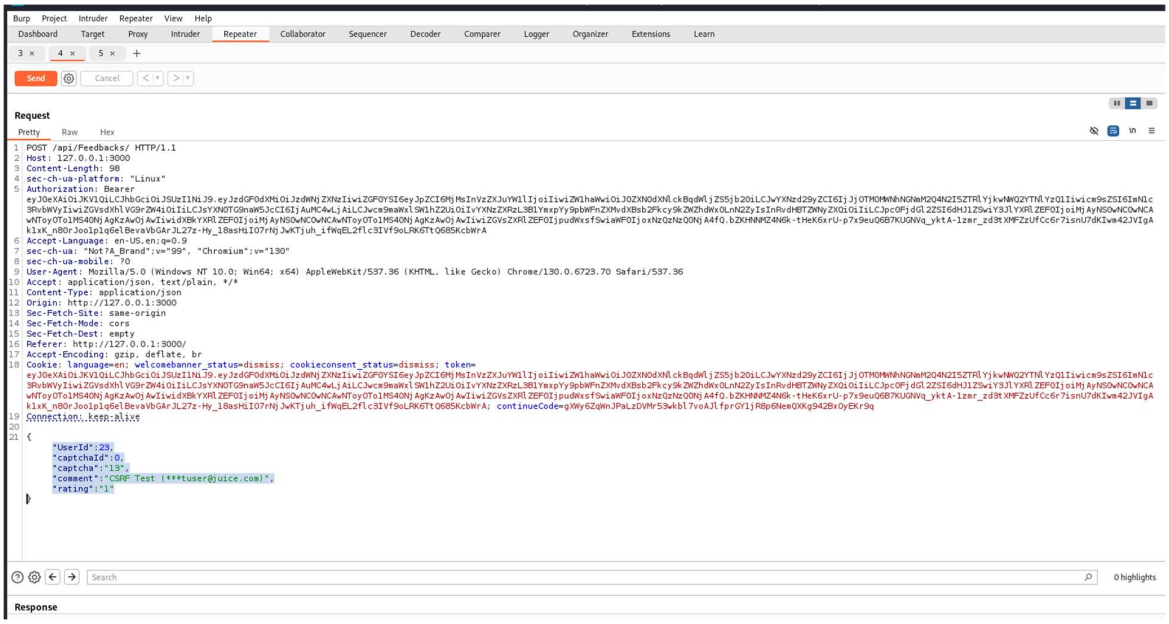
5.1. SQL Injection (SQLi): Injected SQL payloads into the login API using Burp Suite Repeater to test if user input was properly validated.



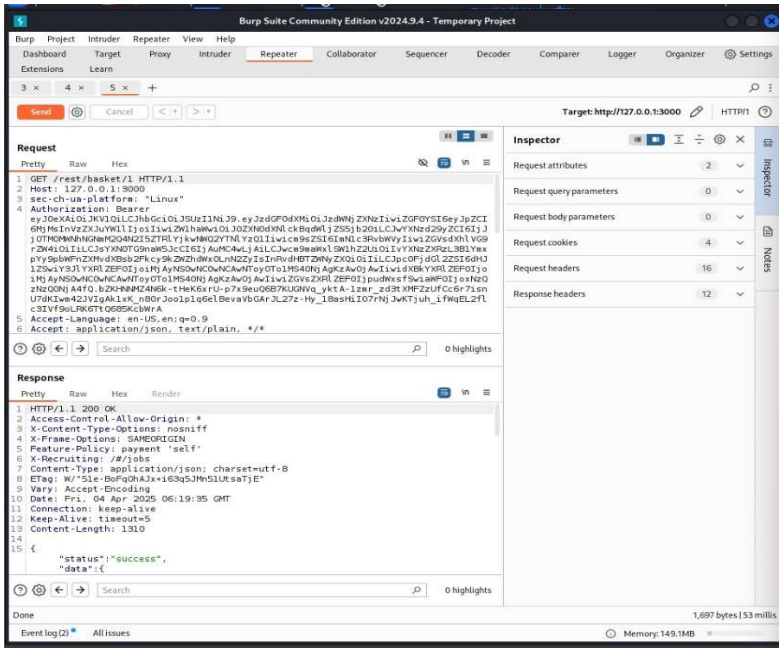
5.2. Brute Force Attack: Used Burp Suite Intruder to automate multiple login attempts, testing if the app lacked rate limiting or lockout mechanisms.



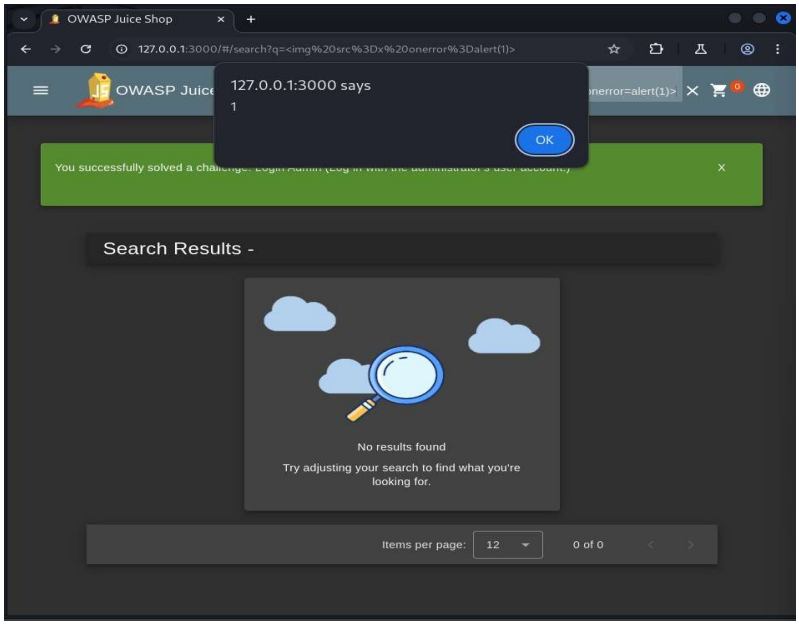
5.3. Cross-Site Request Forgery (CSRF): Intercepted a POST request to /api/Feedback and verified that no CSRF token was being validated, allowing requests to be forged.



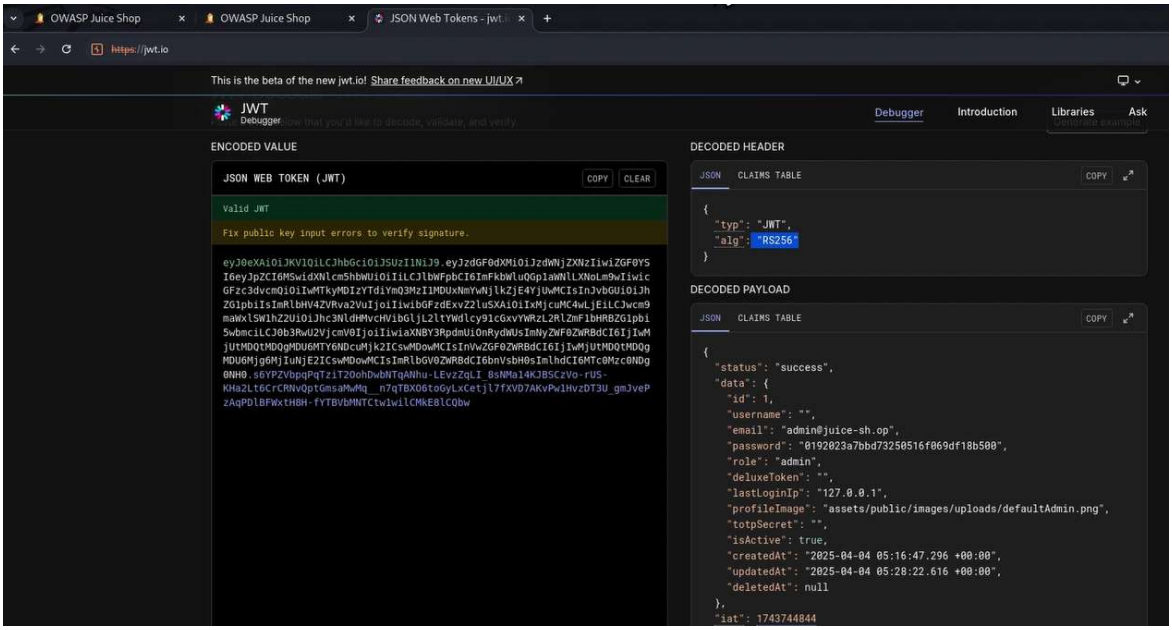
5.4. Insecure Direct Object Reference (IDOR): Modified the basket ID in /rest/basket/:id using Burp Suite Repeater and accessed another user's private cart.



5.5. Cross-Site Scripting (XSS): Injected JavaScript <img src=x onerror=alert(1)> in the search bar and observed its execution, indicating unsanitized input.



5.6. **JSON Web Token (JWT) Misconfiguration:** intercepted tokens during login and account operations using Burp Suite. Verified the structure of the token by decoding its Base64-encoded header and payload. Confirmed that a secure signing algorithm (HS256) was used instead of the insecure none algorithm.



Each vulnerability was discovered by observing how the application responded to unexpected or malicious input and noting the lack of proper security checks.

6. Vulnerabilities Summary.

1. **SQL Injection (SQLi)**
    1. Endpoint Used: /rest/user/login
    2. API Type: POST
    3. Tool Used: Repeater
    4. Confirmed: Yes
    5. Impact: DB Access
    6. This vulnerability was confirmed by injecting SQL payloads into the login API to test if user input was properly validated. Successful exploitation could provide attackers with access to the database.

2. **Brute Force**
    1. Endpoint Used: /rest/user/login
    2. API Type: POST
    3. Tool Used: Intruder
    4. Confirmed: Yes
    5. Impact: Unauthorized Access
    6. The brute force attack was conducted to check for the lack of rate limiting or account lockout mechanisms. Successful attacks allowed unauthorized login attempts, potentially compromising user accounts.

3. **Cross-Site Request Forgery (CSRF)**
    1. Endpoint Used: /api/Feedback
    2. API Type: POST
    3. Tool Used: Repeater, Terminal
    4. Confirmed: Yes
    5. Impact: Forged Actions
    6. The absence of CSRF token validation was confirmed by intercepting a POST request to the feedback endpoint. This vulnerability allowed attackers to forge actions on behalf of authenticated users.

4. **Insecure Direct Object Reference (IDOR)**
    1. Endpoint Used: /rest/basket/:id
    2. API Type: GET
    3. Tool Used: Repeater
    4. Confirmed: Yes
    5. Impact: Data Exposure
    6. The IDOR vulnerability was identified by modifying the basket ID in the API request, allowing access to another user's private cart. This exposes sensitive user data.

5. **Cross-Site Scripting (XSS)**
    1. Endpoint Used: /rest/products/search
    2. API Type: GET
    3. Tool Used: Browser/Repeater
    4. Confirmed: Yes
    5. Impact: Script Injection
    6. XSS was confirmed by injecting a malicious JavaScript payload into the search field, resulting in the execution of the script, which could be used for session hijacking or phishing attacks.

6. **JWT Misconfiguration**

    1. Endpoint Used: /rest/user/login
    2. API Type: POST
    3. Tool Used: Burp Suite Intruder, Decoder
    4. Confirmed: No Critical Flaw
    5. Impact: Potential Unauthorized Access if Misconfigured
    6. The JWT token used a secure signing algorithm (HS256), and no none algorithm was accepted. Signature verification was correctly enforced. The test helped confirm that token validation mechanisms were in place. However, consistent 200 responses for tampered tokens highlight the importance of detailed response analysis in JWT auditing.

7. Impact of the Problem If It Is Not Solved

If REST API vulnerabilities are not properly identified and mitigated, they can lead to serious consequences. Below are the key risks associated with unaddressed REST API security flaws:

1. **SQL Injection (SQLi)**
   Attackers can manipulate database queries through user input, allowing them to extract, alter, or delete sensitive data, leading to complete database compromise.
2. **Brute Force Attacks**
   Without mechanisms like rate limiting or account lockout, attackers can perform repeated login attempts to guess user credentials and gain unauthorized access.
3. **Cross-Site Request Forgery (CSRF)**
   If CSRF tokens are not implemented, attackers can trick authenticated users into unknowingly executing actions on their behalf, compromising account integrity.
4. **Insecure Direct Object Reference (IDOR)**
   Improper authorization checks allow attackers to access or modify data that belongs to other users simply by manipulating object references like user IDs or basket IDs.
5. **Cross-Site Scripting (XSS)**
   Unsanitized input fields can be exploited to inject malicious scripts, leading to session hijacking, phishing attacks, and theft of sensitive user information.

6. **Loss of Customer Trust and Reputation Damage**
   Security breaches undermine user confidence, potentially leading to a loss of customers, damage to brand reputation, and reduced market competitiveness.
7. **Regulatory Non-Compliance**
   Organizations may face legal action, fines, and sanctions if they fail to comply with data protection laws such as GDPR or HIPAA due to poor API security.

8. Mitigation Recommendations

Brief countermeasures for each vulnerability:

1. **SQLi:** Use parameterized queries.
2. **Brute Force:** Implement rate limiting, CAPTCHA.
3. **CSRF:** Use CSRF tokens and SameSite cookies.
4. **IDOR:** Enforce access control on every API.
5. **XSS:** Sanitize and encode user input/output.
6. **JWT**: Always validate the JWT signature on the backend, use strong signing algorithms like HS256/RS256, and avoid accepting tokens signed with none. Implement token expiry checks and secure storage.

9. Conclusion

In this project, a security audit of the OWASP Juice Shop's REST APIs was conducted, identifying critical vulnerabilities such as SQL Injection, Brute Force Attacks, CSRF, IDOR, and XSS. These vulnerabilities highlight the need for robust security measures, including parameterized queries, rate limiting, CSRF tokens, access controls, and input sanitization. The JWT implementation was also reviewed to ensure secure signing algorithms and proper token validation were in place, reducing the risk of unauthorized access via token manipulation. The findings emphasize the importance of securing APIs to protect user data and ensure application integrity. The project serves as a valuable guide for developers to implement secure API practices and prevent potential exploits.

10. References

1. OWASP Foundation. "OWASP Juice Shop." https://owasp.org/www-project-juice-shop/
2. Burp Suite. "Burp Suite Documentation." https://portswigger.net/burp/documentation
3. OWASP Foundation. "OWASP Top Ten." https://owasp.org/www-project-top-ten/
4. Gupta, A., & Kaur, H. (2020). "A Comprehensive Study of Web Application Security Testing Tools." *International Journal of Computer Applications*, 176(4), 5-9.
5. SANS Institute. "Web Application Security: An Introduction." https://www.sans.org/cyber-security-courses/web-application-security/
6. OWASP Foundation. "OWASP Cheat Sheet Series." https://cheatsheetseries.owasp.org/