TASK-2:“Understanding and Analyzing Threat Models”

Threat Modeling Report for OWASP Juice Shop

1. Introduction

As part of my internship task, I was asked to perform threat modeling on a vulnerable application. I selected OWASP Juice Shop, which is considered one of the most insecure web applications intentionally created for learning purposes. Juice Shop is built on modern web technologies and contains a wide range of security flaws such as SQL Injection, Cross-Site Scripting (XSS), Insecure Direct Object References, Broken Authentication, and more.

The purpose of this document is to analyze the application using a structured threat modeling approach, identify possible security threats, and recommend mitigation strategies.

2. Threat Modeling Approach

For this document, I used the STRIDE model developed by Microsoft. STRIDE is a mnemonic that helps to categorize different types of threats. Each letter represents a specific category of security concern:

- S – Spoofing Identity
- T – Tampering with Data
- R – Repudiation
- I – Information Disclosure
- D – Denial of Service (DoS)
- E – Elevation of Privilege

Using STRIDE, I was able to systematically evaluate the potential risks in Juice Shop and map them to relevant controls.

3. System Overview of Juice Shop

OWASP Juice Shop is a demo e-commerce web application that allows users to register, log in, browse products, place orders, and manage accounts. It has several main components:

- Frontend (Angular/JavaScript) – user interface for shopping and account management
- Backend (Node.js/Express) – handles business logic and API requests
- Database (SQLite/Postgres) – stores user accounts, products, and orders
- Authentication Module – for login and session management
- Admin Panel – for privileged management activities

This makes Juice Shop a good candidate for threat modeling as it resembles a real-world online shopping platform.

4. STRIDE Threat Analysis

4.1 Spoofing Identity

Attackers attempt to impersonate other users or gain unauthorized access. Example in Juice Shop:

- Weak login system that can be bypassed using SQL Injection (e.g., entering `' OR 1=1 --`).
- Session tokens stored in local storage can be stolen and reused.

Mitigation:

- Use parameterized queries to prevent SQL Injection.
- Implement secure session management (HttpOnly, Secure cookies, short expiry).
- Enforce multi-factor authentication (MFA).

4.2 Tampering with Data

Modifying or manipulating data without authorization.
Example in Juice Shop:

- Changing the price of items in client-side requests before sending them to the server.
- Modifying hidden fields in HTML to get discounts or free products.

Mitigation:

- Never trust client-side input; validate on server-side.
- Use hashing or digital signatures for sensitive data.
- Apply input/output validation at every layer.

4.3 Repudiation

A user denies performing an action without sufficient audit logs to prove otherwise.
Example in Juice Shop:

- A malicious user deletes data or makes an unauthorized purchase and later claims they did not.

Mitigation:

- Implement comprehensive logging for all sensitive operations.
- Use secure audit trails that cannot be tampered with.
- Add time-stamped logs with user IDs.

4.4 Information Disclosure

Sensitive data is leaked to unauthorized users.
Example in Juice Shop:

- Error messages displaying full SQL queries.

- Exposed API endpoints that reveal customer data.
- Sensitive information stored in plaintext (e.g., credit card numbers).

Mitigation:

- Show generic error messages instead of detailed ones.
- Encrypt sensitive data both in storage (AES) and transit (TLS/SSL).
- Implement access control checks for APIs.

4.5 Denial of Service (DoS)

Making the application unavailable to legitimate users.
Example in Juice Shop:

- Sending thousands of fake requests to overload the server.
- Exploiting poorly optimized queries to slow down the database.

Mitigation:

- Implement rate limiting and request throttling.
- Use Web Application Firewall (WAF) to filter malicious requests.
- Optimize database queries and implement caching.

4.6 Elevation of Privilege

A user gains higher privileges than intended.
Example in Juice Shop:

- A normal user accessing the /admin panel by manipulating cookies or session tokens.
- Exploiting insecure direct object references (IDOR) to perform admin actions.

Mitigation:

- Apply role-based access control (RBAC).

- Validate permissions server-side before executing sensitive actions.
- Secure APIs with authorization checks.

6. Security Controls Summary

STRIDE Category	Potential Threat in Juice Shop	Security Controls
Spoofing	SQL Injection login bypass	Parameterized queries, MFA
Tampering	Changing product prices in requests	Server-side validation, input sanitization
Repudiation	Denying unauthorized actions	Logging, audit trails
Information Disclosure	Exposed error messages and APIs	Generic errors, encryption
DoS	Fake requests overload	Rate limiting, WAF
Elevation of Privilege	Unauthorized admin panel access	RBAC, server-side checks

7. Conclusion

Through this exercise, I understood how threat modeling helps in identifying potential security risks in applications before they are exploited. Using the STRIDE approach, I was able to analyze OWASP Juice Shop and classify threats into six categories. For each threat, I also mapped possible mitigations.

This activity helped me to think like both an attacker and a defender. As a student, this was a very valuable learning experience because it gave me practical exposure to real-world security concerns. If such vulnerabilities existed in a real e-commerce application, they could cause financial loss, data breaches, and reputational damage.

By applying proper security controls, developers can make Juice Shop (and similar applications) much safer for end users.