**Tasks  
Task 1: Research common threat models (STRIDE, PASTA, etc.).**

* **STRIDE (Microsoft)**

STRIDE identifies threats under six categories:

* **S**poofing identity
* **T**ampering with data
* **R**epudiation
* **I**nformation disclosure
* **D**enial of service
* **E**levation of privilege
* **PASTA (Process for Attack Simulation and Threat Analysis)**

PASTA is a 7-step risk-centric methodology:

1. Define business objectives
2. Define technical scope
3. Decompose the application
4. Analyze threats
5. Vulnerability analysis
6. Attack modeling
7. Risk and impact analysis.

**Task 2: Document a threat model for your vulnerable application.**

## Threat Model for OWASP Juice Shop (Using STRIDE)

This document presents a threat model for the OWASP Juice Shop application using the STRIDE methodology. It identifies potential security threats, affected components, and proposes mitigation strategies to improve the application's security posture.

**App Name**: OWASP Juice Shop  
**Methodology Used**: STRIDE

### ****1. Application Overview****

**Description**:  
OWASP Juice Shop is a modern single-page application (SPA) built using Node.js (Express), Angular, and SQLite. It simulates a real e-commerce site and includes common vulnerabilities intentionally.

**Architecture**:

* **Frontend**: Angular SPA
* **Backend**: Node.js with Express
* **Database**: SQLite
* **Deployment**: Localhost or Docker, also available on Heroku & other platforms

### ****2. Assets to Protect****

* User credentials (usernames, passwords, tokens)
* Payment and personal information
* Admin interface
* Product catalog and pricing
* Order data

### ****3. Data Flow Structure (DFS)****

**Actors**:

* End User
* Attacker
* Admin

**Key Components**:

* Browser (Client)
* Angular App (Frontend)
* REST API (Backend)
* Database

**Typical Flows**:

1. Login Request → API → Auth Service → DB
2. Browse Products → API → DB
3. Place Order → API → DB
4. Admin Panel Access → API → DB

**4.STRIDE Threat Analysis**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **STRIDE Category** | **Threat Example** | **Vulnerable Component** | **Mitigation** | | **S**poofing | Login without password or brute-force login | Login endpoint | Use rate-limiting, CAPTCHA, strong password policies | | **T**ampering | Modify pricing via client-side JS | Frontend/API | Use server-side validation, secure API | | **R**epudiation | User deletes order history or performs action without logs | Orders API | Enable logging and audit trails | | **I**nformation Disclosure | Access other users’ orders via IDOR | Order API | Implement proper authorization and access control | | **D**enial of Service | Repeated API calls crash backend | API | Rate limiting, circuit breakers | | **E**levation of Privilege | Regular user accesses admin panel | Admin interface | Use RBAC and token-based access control | |

### ****5. Recommended Security Control****

* Implement **rate limiting** on sensitive endpoints
* Enforce **input validation and output encoding**
* Use **OAuth2/JWT** for secure token-based auth
* Enable **audit logging** for user actions
* Apply **Content Security Policy (CSP)** headers
* Enforce **secure coding practices** for Angular & Node.js

**Task 3: Identify Potential Threats and Security Controls**

This section identifies potential security threats in the OWASP Juice Shop application using the STRIDE model. For each threat category, related vulnerabilities are analyzed and appropriate security controls are recommended to reduce risk and improve overall security.

* **S**poofing
* **T**ampering
* **R**epudiation
* **I**nformation Disclosure
* **D**enial of Service (DoS)
* **E**levation of Privilege

**STRIDE Threat Breakdown for OWASP Juice Shop**

**1. Spoofing Identity**

* **Threat**: An attacker impersonates a legitimate user (e.g., logging in without correct credentials).
* **Vulnerable Components**: Login page, token handling, password reset.
* **Potential Attacks**:
  + Credential stuffing
  + Brute-force login
  + Session hijacking
* **Security Controls**:
  + Multi-Factor Authentication (MFA)
  + Strong password policies
  + Account lockout after repeated failures
  + Secure session tokens (HttpOnly, Secure, SameSite)

**2. Tampering with Data**

* **Threat**: An attacker modifies data in transit or at rest.
* **Vulnerable Components**: Product pricing, shopping cart, API requests.
* **Potential Attacks**:
  + Changing prices using browser dev tools
  + Modifying JWTs or cookies
  + Tampering with API payloads (e.g., modifying order quantities)
* **Security Controls**:
  + Server-side input validation (never trust client-side values)
  + Digital signatures (e.g., signed JWTs)
  + Use of HTTPS to protect data in transit
  + Integrity checks and hashing

**3. Repudiation**

* **Threat**: A user denies performing an action, and there's no way to prove otherwise.
* **Vulnerable Components**: Order history, payment activity, user actions.
* **Potential Attacks**:
  + User claims they didn’t place an order
  + Malicious actor deletes logs or alters actions
* **Security Controls**:
  + Implement detailed logging with timestamps and user IDs
  + Store logs in tamper-evident systems (e.g., centralized logging with immutability)
  + Monitor logs regularly for anomalies

**4. Information Disclosure**

* **Threat**: Unauthorized access to sensitive data.
* **Vulnerable Components**: APIs, database queries, error messages.
* **Potential Attacks**:
  + Insecure Direct Object References (IDOR) revealing other users’ data
  + Stack traces or sensitive data shown in error messages
  + Leaky APIs revealing admin emails or passwords
* **Security Controls**:
  + Enforce Access Control on every API route
  + Sanitize and customize error messages (e.g., 403 instead of stack trace)
  + Use proper authorization checks (e.g., only return orders for logged-in user)

**5. Denial of Service (DoS)**

* **Threat**: Preventing legitimate users from accessing the app.
* **Vulnerable Components**: Search, login, review posting.
* **Potential Attacks**:
  + Spamming search API with heavy queries
  + Flooding review or registration forms
  + Crashing server with malformed payloads
* **Security Controls**:
  + Implement rate limiting (e.g., 5 logins per minute)
  + Use Web Application Firewalls (WAF)
  + Validate input to prevent payload abuse
  + Queue heavy operations asynchronously

**6. Elevation of Privilege**

* **Threat**: A user gains higher access rights than intended.
* **Vulnerable Components**: Admin dashboard, role management, hidden functions.
* **Potential Attacks**:
  + Changing JWT role from "user" to "admin"
  + Accessing hidden admin routes via URL manipulation
  + Bypassing client-side access restrictions
* **Security Controls**:
  + Implement robust Role-Based Access Control (RBAC) on the backend
  + Never rely on frontend for access restrictions
  + Secure JWTs and validate all claims server-side
  + Audit user role changes

|  |  |  |
| --- | --- | --- |
| **STRIDE** | **Threat Example** | **Controls** |
| **Spoofing** | Credential stuffing | MFA, account lockout, secure tokens |
| **Tampering** | Modify cart prices | Server-side validation, signed tokens |
| **Repudiation** | User denies actions | Logging, audit trails |
| **Information Disclosure** | Exposed user data | Access control, error sanitization |
| **DoS** | API spam | Rate limiting, input validation |
| **Privilege Escalation** | User accesses admin panel | RBAC, server-side enforcement |