Risk Assessment and Mitigation Pan

Purpose

This document identifies potential risks in the booking system, evaluates their severity, proposes mitigation strategies, and highlights residual risks. It serves as a foundational guide to secure the application while balancing feasibility and scope.

Risk Assessment Table

Risk	Severity (L/M/H)	Mitigation	Residual Risk
Injection Attacks (SQL/NoSQL)	High	Use parameterized queries, sanitize inputs, and implement input validation at all entry points.	Minimal with proper sanitization.
Cross-Site Scripting (XSS)	High	Escape user-generated content in the UI, implement CSP, and use libraries like DOMPurify.	Some residual risk with new XSS techniques.
Broken Authentication	High	Use JWT with short expiration, implement MFA, and secure token storage in `HttpOnly` cookies.	Token theft or session hijacking remains a minor residual risk.
Session Hijacking	Medium	Use secure cookies, implement session timeout, and enforce HTTPS for all communications.	Residual risk if a user's device is compromised.
Sensitive Data Exposure	High	Enforce TLS 1.2 or higher, use strong password hashing (bcrypt), and store sensitive data securely.	Minimal if encryption and access controls are strictly followed.
Insecure Deserialization	Medium	Validate and sanitize all serialized data, and avoid insecure serialization formats.	Residual risk with third-party library use.
Lack of Access Control	High	Implement RBAC, validate permissions on all backend endpoints, and audit logs for unauthorized access.	Residual risk due to potential misconfiguration.
Insufficient Logging and Monitoring	Medium	Implement centralized logging with tools like Winston, monitor critical events, and set up alerts.	Potential gaps in real-time response for rare incidents.
Denial-of-Service (DoS)	Medium	Rate-limit API endpoints, use CAPTCHAs, and employ WAF rules for known attack patterns.	Some residual risk from large-scale botnets.

Supply Chain Attacks	Medium	Regularly update dependencies, verify integrity with hashes, and monitor security advisories.	Residual risk with newly discovered vulnerabilities in dependencies.

High-Risk Areas and Prioritized Actions

1. **Injection Attacks**:
    a. **Action**: Prioritize input sanitization and the use of ORM libraries (e.g., Mongoose for MongoDB).
    b. **Justification**: Injection attacks can compromise the database and sensitive user information.
2. **Broken Authentication**:
    a. **Action**: Implement MFA, secure JWT handling, and enforce strong password policies.
    b. **Justification**: Authentication vulnerabilities could allow attackers to impersonate users or admins.
3. **Sensitive Data Exposure**:
    a. **Action**: Enforce HTTPS, use strong encryption for stored passwords (`bcrypt`), and sanitize logs to avoid leaking sensitive data.
    b. **Justification**: Protects user and application data from interception or leaks.
4. **Access Control**:
    a. **Action**: Implement RBAC and validate permissions on sensitive routes (e.g., `/api/rooms`, `/api/bookings`).
    b. **Justification**: Prevent unauthorized access to critical resources.
5. **Cross-Site Scripting (XSS)**:
    a. **Action**: Escape and sanitize all user-generated content before rendering in the UI.
    b. **Justification**: Prevent attackers from injecting malicious scripts.

Residual Risks

1. **Advanced Injection Techniques**:
    a. Even with input sanitization, new or unknown injection techniques may emerge.
    b. **Threat Actor**: Skilled attackers using automated tools.
    c. **Mitigation**: Regularly update frameworks and monitor security advisories.
2. **Token Theft or Session Hijacking**:
    a. Despite `HttpOnly` cookies and HTTPS, attackers might exploit user-side vulnerabilities.
    b. **Threat Actor**: Users at public Wi-Fi hotspots or phishing campaigns.
    c. **Mitigation**: Encourage users to log out after use and avoid insecure networks.
3. **Denial-of-Service (DoS)**:
    a. Large-scale botnets could overwhelm the server despite rate-limiting.
    b. **Threat Actor**: Script kiddies, competitors, or students experimenting with DoS tools.
    c. **Mitigation**: Consider cloud-based solutions (e.g., AWS WAF) for additional scalability.
4. **Compromised Dependencies**:
    a. New CVEs in third-party libraries might go undetected until exploited.
    b. **Threat Actor**: Hackers exploiting known CVEs in Node.js packages.
    c. **Mitigation**: Automate dependency checks with tools like `npm audit`.
5. **Potential Internal Threats**:
    a. Students or users with legitimate access might attempt to bypass restrictions or exploit vulnerabilities.
    b. **Mitigation**: Audit logs frequently and consider IP whitelisting for admin access.

Summary of Key Mitigations

- Address high-risk areas such as authentication and input validation first.
- Regularly monitor and patch vulnerabilities in dependencies and frameworks.
- Enforce access controls and RBAC to minimize unauthorized access.
- Integrate logging and alerting to detect and respond to potential breaches quickly.
- Educate end-users about safe practices, especially when using public networks or sharing credentials.

- End -