Threat Model and Risk Assessment

1. Introduction

Purpose of the Document

The purpose of this Threat Model Document is to identify, analyze, and propose mitigations for potential security threats to the booking system. This document aims to ensure that the system's core assets—such as user data, booking data, and authentication mechanisms—are protected from malicious attacks, unauthorized access, and potential vulnerabilities. By adopting a proactive security stance, we aim to reduce the likelihood and impact of threats, thereby increasing the security and reliability of the booking system.

Scope

The scope of this threat model includes the entire booking system architecture, encompassing the frontend, backend (server and API), database, and supporting components such as authentication, booking management, and role-based access control (RBAC). It does not extend to third-party dependencies, though we assume they follow industry-standard security practices.

Assumptions

- Users interact with the system via a secure web browser.
- Access to the backend server and database is controlled and limited to authorized personnel only.
- Users and administrators are authenticated before accessing restricted functionalities.
- Multi-Factor Authentication (MFA) is available for enhanced security.
- Sensitive data in transit is encrypted.

2. System Overview

The booking system is a web application built using the MERN stack (MongoDB, Express.js, HTML/CSS/JS, Node.js). It allows users to book rooms by selecting available time slots, view their booking history, and receive booking confirmations. Administrators have the ability to manage rooms, view all bookings, and perform administrative tasks through a secure admin dashboard.

The system architecture comprises the following components:

- **Frontend (Client Side)**: Built with HTML, CSS, and JavaScript. It provides a user-friendly interface for users to interact with the booking system.
- **Backend (Server Side)**: Developed using Node.js and Express.js. This layer handles business logic, processes HTTP requests, and interacts with the MongoDB database.
- **Database**: MongoDB is used to store information on users, rooms, and bookings. The database holds critical information, such as user credentials, booking details, and room availability.
- **Authentication Module**: JWT tokens and Role-Based Access Control (RBAC) are used to authenticate users and authorize access to different functionalities. Multi-Factor Authentication (MFA) is also implemented to provide additional security for user accounts.
- **Controllers**: Specialized modules in the backend that handle CRUD operations for rooms and bookings. They are responsible for processing data and enforcing business rules.
- **Security Middleware**: Custom middlewares manage authentication, enforce RBAC, and log or block suspicious activity to help mitigate unauthorized access and other security risks.

Data Flow

1. Users access the booking system through a web client and interact with the frontend components.
2. HTTP requests (such as booking a room, viewing room details) are sent to the backend server.
3. The backend processes requests, manages authentication and authorization, and interacts with the MongoDB database for CRUD operations.
4. Booking confirmations, booking history, and room availability data are relayed back to the frontend for user interaction.

3. Assets and Security Goals

Assets

Key assets within the booking system are as follows:

- **User Data**:
  - User credentials (username, password)
  - Personal information (email addresses, usernames)
  - Authentication tokens (JWTs) and MFA secrets
- **Room Data**: Information about each room, such as location, capacity, amenities, and availability status.
- **Booking Data**: Details of each booking, including room ID, booking date, time slot, and user ID.
- **Admin and User Privileges**: Role-based access control allows specific users (admins) to manage rooms, view all bookings, and perform privileged actions.
- **Application Logic and Configuration**: This includes API endpoints, business rules, and environment configurations used within the backend.

Security Goals

The primary security goals for the booking system are defined as follows:

- **Confidentiality**:
  - Ensure that only authenticated users and administrators can access sensitive data.
  - Encrypt sensitive data, such as passwords and authentication tokens, to prevent unauthorized access.
- **Integrity**:
  - Ensure that booking and room data cannot be tampered with by unauthorized users.
  - Validate user input to prevent injection attacks and ensure data integrity.
- **Availability**:
  - Ensure that the booking system remains available and responsive for legitimate users.
  - Prevent denial-of-service attacks by implementing rate limiting and resource management practices.
- **Authentication and Authorization**:
  - Require users to authenticate before accessing the system.
  - Use RBAC to ensure that only authorized users have access to administrative functionalities.
  - Implement MFA for an additional layer of security on user accounts.
- **Accountability and Traceability**:
  - Maintain logs of all sensitive actions, such as room management and booking cancellations, for auditing purposes.
  - Enable repudiation controls to ensure actions can be traced back to a specific user or admin, enhancing accountability.

By meeting these goals, the booking system aims to safeguard its assets and maintain a secure, reliable experience for student and admin users.

4. Threat Identification

Threat Modeling Approach: STRIDE

The following threats have been identified by applying the STRIDE framework to the system's primary components: **Client Side (Frontend)**, **Server Side (Backend)**, **Database (MongoDB)**, **Authentication Module**, and **Controllers**. This threat assessment will be essential for prioritizing security measures and mitigating risks in the booking system.

4.1 Client Side (Frontend)

Threat Type	Threat Description
Spoofing	Attackers may attempt to impersonate legitimate users by exploiting weaknesses in the login or session management process on the client side. This could occur through phishing or by stealing authentication tokens stored in local storage.
Tampering	Users with access to developer tools may attempt to manipulate client-side data (e.g., booking details, price values) before sending it to the server, potentially altering requests and gaining unauthorized advantages.
Repudiation	Without proper logging, users could deny their actions (such as creating or canceling bookings) because of insufficient audit trails on the frontend.
Information Disclosure	Sensitive data, such as JWT tokens or personal information, stored in local storage could be accessed by attackers if the browser or client device is compromised.
Denial of Service (DoS)	Attackers could overload the frontend with automated requests, potentially leading to degraded performance or the system becoming unavailable to legitimate users.
Elevation of Privilege	Users may attempt to escalate privileges by tampering with the client-side code to access restricted features, such as administrative functions.

4.2 Server Side (Backend)

Threat Type	Threat Description
Spoofing	Attackers may attempt to impersonate administrators by using stolen JWT tokens or weak session management protocols to access restricted endpoints.
Tampering	Attackers may intercept and modify HTTP requests in transit, attempting to alter booking data or perform actions beyond their authorization level.
Repudiation	If detailed logging is not implemented, attackers may exploit this to obscure their activities and deny actions, making it difficult to trace malicious activities.
Information Disclosure	Unsecured endpoints could potentially leak sensitive information, such as user details, room availability, and booking data, if they are improperly configured.
Denial of Service (DoS)	Attackers could launch a DoS attack on the server by flooding it with requests, which could disrupt service for legitimate users.
Elevation of Privilege	Attackers may attempt to escalate their privileges by exploiting vulnerabilities in the server code, such as weak access controls on certain endpoints.

4.3 Database (MongoDB)

Threat Type	Threat Description
Spoofing	Attackers may attempt to impersonate the database connection through configuration tampering or misconfigured credentials to access and manipulate data.
Tampering	Attackers who gain unauthorized access to the database could modify user or booking data, such as changing booking statuses, altering room availability, or tampering with user roles.
Repudiation	Lack of database-level logging could allow attackers to manipulate data without leaving a trace, making it hard to hold individuals accountable for unauthorized changes.
Information Disclosure	Unauthorized users could gain access to sensitive data if the database permissions are misconfigured or if the database is exposed to the internet without proper security.
Denial of Service (DoS)	A database DoS attack, such as a large number of queries, could degrade performance, leading to slowdowns or complete outages.
Elevation of Privilege	An attacker who gains access to the database with insufficiently restricted privileges could escalate their access to modify critical data or configurations.

4.4 Authentication Module

Threat Type	Threat Description
Spoofing	Attackers could impersonate users or administrators by stealing their JWT tokens or MFA codes, gaining unauthorized access to the system.
Tampering	Attackers may attempt to manipulate JWT tokens or MFA configuration to bypass authentication and gain unauthorized access.
Repudiation	Without logging failed and successful login attempts, users could deny attempting to access sensitive areas, hindering auditing.
Information Disclosure	If MFA secrets or JWT tokens are improperly stored or transmitted, attackers could intercept these tokens and gain unauthorized access.
Denial of Service (DoS)	An attacker could attempt brute-force login attempts, potentially locking out legitimate users or overwhelming the authentication system.
Elevation of Privilege	Attackers may attempt to escalate privileges by manipulating authentication tokens, allowing unauthorized access to administrative features.

4.5 Controllers (Room Controller, Booking Controller)

Threat Type	Threat Description
Spoofing	Attackers could attempt to spoof requests to the Room or Booking controllers, impersonating administrators or other users to make unauthorized modifications.
Tampering	Attackers could manipulate requests to the Room and Booking controllers, such as changing room availability or booking details, without proper authorization.
Repudiation	If request logs are not maintained, users or administrators could deny actions taken through the controllers, making it challenging to audit system changes.
Information Disclosure	Unauthorized users may access sensitive data about bookings and room configurations if controllers lack proper access controls.
Denial of Service (DoS)	Attackers could flood the controllers with requests, potentially disrupting booking operations and management functions for legitimate users.
Elevation of Privilege	Attackers might attempt to bypass access controls within the controllers to perform actions only allowed for administrators, such as managing rooms and viewing all bookings.

This analysis provides a comprehensive view of potential threats across the booking system components using the STRIDE framework. These identified threats will guide the prioritization of security mitigations and countermeasures in the subsequent sections.

5. Threat Analysis and Mitigation

The following table provides an analysis of each identified threat, including a risk assessment based on **Likelihood** (L) and **Impact** (I) ratings, where both are rated on a scale from 1 (Low) to 5 (High). **Mitigations** are suggested for each threat to reduce the likelihood, impact, or both. The risk score is calculated as **Risk = Likelihood x Impact**.

5.1 Client Side (Frontend)

Threat Type	Description	Likelihood (L)	Impact (I)	Risk (R = L x I)	Mitigations
Spoofing	Impersonation through stolen tokens.	4	4	16	Implement secure cookie storage for tokens, use HTTPS, and add token expiration with regular rotation. Enforce MFA for high-risk actions.
Tampering	Manipulation of client-side data in requests.	3	3	9	Use server-side validation for all requests. Avoid trusting client-provided data without validation. Enable Content Security Policy (CSP) headers.
Repudiation	Users could deny actions due to lack of audit logs.	2	3	6	Implement server-side logging for critical actions. Track actions tied to unique user IDs to establish accountability.
Information Disclosure	Exposure of sensitive data (e.g., tokens) in client storage.	4	4	16	Avoid storing sensitive information in local storage. Use secure cookies with HTTPOnly and Secure flags. Apply data masking where possible.
Denial of Service	Attackers could degrade client performance with repeated requests.	3	3	9	Rate-limit requests to the server and enforce CAPTCHA or reCAPTCHA for high-frequency actions. Cache static resources to reduce load on the client.
Elevation of Privilege	User attempts to access restricted functions by altering client-side code.	3	4	12	Enforce role-based access control (RBAC) on the server side. Hide admin-only features on the client side and verify permissions server-side.

5.2 Server Side (Backend)

Threat Type	Description	Likelihood (L)	Impact (I)	Risk (R)	Mitigations
Spoofing	Attackers may impersonate admins via stolen tokens.	3	5	15	Enforce MFA for admin actions. Implement token expiration, rotation, and secure storage. Log admin access and alert on suspicious patterns.
Tampering	Attackers could alter HTTP requests in transit.	3	4	12	Use HTTPS to encrypt traffic. Validate and sanitize all incoming requests. Use HMAC signatures to verify request integrity.
Repudiation	Lack of detailed logging may obscure malicious actions.	2	4	8	Implement comprehensive server-side logging with unique identifiers for each action. Store logs securely and ensure only authorized access.
Information Disclosure	Unsecured endpoints could leak sensitive data.	3	5	15	Limit data exposure through API responses. Use encryption at rest and in transit. Secure sensitive endpoints with authentication and authorization checks.
Denial of Service	Attackers could overload the server with requests.	4	4	16	Implement rate limiting, load balancing, and WAF rules to mitigate DoS. Consider IP blacklisting for repeat offenders.
Elevation of Privilege	Attackers could exploit server vulnerabilities to escalate privileges.	3	5	15	Conduct regular vulnerability scans and penetration testing. Implement strict RBAC, restrict access to sensitive operations, and regularly review permissions.

5.3 Database (MongoDB)

Threat Type	Description	Likelihood (L)	Impact (I)	Risk (R)	Mitigations
Spoofing	Impersonation of database connection through tampered credentials.	2	5	10	Restrict database access to specific IPs. Use secure passwords and avoid hardcoding credentials. Enable MongoDB authentication and TLS.
Tampering	Unauthorized access could allow modification of sensitive data.	3	5	15	Enforce database access control with RBAC. Log all database changes and restrict access based on roles. Enable audit trails.
Repudiation	Lack of database logs may allow users to deny actions.	2	4	8	Enable audit logging in MongoDB. Configure logs to capture important actions and modifications. Limit access to logs to authorized users.
Information Disclosure	Misconfigured permissions could expose sensitive data.	3	5	15	Limit user roles and permissions in the database. Use encryption for sensitive fields (e.g., user credentials). Enable data access monitoring.
Denial of Service	Attackers could flood the database with queries.	4	4	16	Set up rate limits and configure timeouts for database queries. Enable auto-scaling for high-load periods. Use WAF to filter malicious requests.
Elevation of Privilege	Unauthorized access to database roles could escalate privileges.	2	5	10	Implement strict RBAC within MongoDB. Review and minimize database permissions periodically. Use strong authentication and secure database connections.

5.4 Authentication Module

Threat Type	Description	Likelihood (L)	Impact (I)	Risk (R)	Mitigations
Spoofing	Attackers impersonate users/admins by stealing tokens or MFA codes.	4	5	20	Enforce MFA for all users. Implement short-lived tokens with secure storage. Use IP-based or device-based access monitoring.
Tampering	Attackers manipulate JWT tokens to bypass authentication.	3	5	15	Use HMAC or RSA signatures for tokens and validate them on the server. Rotate signing keys periodically and apply strict token validation.
Repudiation	Users may deny login attempts without sufficient logging.	3	4	12	Log all login attempts with timestamp, IP address, and device info. Alert on multiple failed attempts. Store logs securely.
Information Disclosure	MFA secrets or JWT tokens may be exposed if not securely stored.	3	5	15	Store MFA secrets and tokens securely (e.g., secure cookies). Enforce HTTPS and avoid exposing secrets in URLs or logs.
Denial of Service	Brute-force attempts may overwhelm the authentication module.	4	4	16	Rate-limit login attempts and enforce CAPTCHA after multiple failures. Set account lockouts for excessive login attempts.
Elevation of Privilege	Attackers escalate privileges by tampering with tokens.	2	5	10	Use RBAC and validate all actions based on roles. Log and monitor access to privileged actions and review roles regularly.

5.5 Controllers (Room Controller, Booking Controller)

Threat Type	Description	Likelihood (L)	Impact (I)	Risk (R)	Mitigations
Spoofing	Attackers attempt to impersonate legitimate users to interact with controllers.	3	4	12	Enforce strong session management, token-based authentication, and access controls. Log and monitor all critical actions.
Tampering	Unauthorized modification of requests to manipulate booking/room data.	4	4	16	Validate all incoming data server-side. Implement strict input sanitization and enforce RBAC on controller actions.
Repudiation	Users could deny actions due to insufficient logging.	2	3	6	Implement detailed logging for all critical actions with unique request IDs. Retain logs securely and ensure they're only accessible to authorized personnel.
Information Disclosure	Sensitive room or booking details may be disclosed via unsecured endpoints.	3	4	12	Ensure all sensitive data is accessible only to authorized users. Secure endpoints and restrict access based on roles. Apply data masking where appropriate.
Denial of Service	Attackers could overwhelm controllers with repeated requests.	4	4	16	Apply rate limiting, caching for frequent requests, and circuit breakers to prevent overload. Monitor for patterns of abnormal activity.
Elevation of Privilege	Attackers could attempt to escalate access to controller actions by tampering with requests.	3	5	15	Implement role-based access checks within controllers. Limit controller access based on roles and ensure proper authorization for each request.

This Threat Analysis and Mitigation section outlines potential threats across the main system components and provides a prioritized list of mitigations for each identified threat.

This approach helps in systematically reducing the security risks and enhancing the overall security posture of the booking system.

6. Residual Risks

After implementing the mitigations outlined in the Threat Analysis and Mitigation section, certain risks may still persist due to the inherent complexity of web applications, limitations in available technology, and potential zero-day vulnerabilities. This section lists these **residual risks** along with possible CVEs, threat actors, and the motivation of specific groups such as cybersecurity students who may attempt to test the application's defenses.

6.1 Residual Risks

1. **Advanced Spoofing Attacks on Authentication**
    a. **Description**: Despite implementing multi-factor authentication (MFA) and JWT tokens, sophisticated attackers could still employ social engineering or phishing techniques to gain access to user credentials.
    b. **Mitigation Limitations**: While MFA helps, it is not foolproof against highly targeted social engineering attacks, which could trick users into disclosing their OTP codes.
    c. **Associated CVEs**: CVE-2021-20016 (related to credential spoofing vulnerabilities in web applications).
    d. **Threat Actors**: Experienced hackers or students with a focus on penetration testing may attempt to employ social engineering or spoofing techniques.

2. **Tampering via Man-in-the-Middle (MITM) Attacks**
    a. **Description**: Although HTTPS is enforced to protect data in transit, users connected to insecure networks (e.g., public Wi-Fi) may still be vulnerable to sophisticated MITM attacks, especially if DNS hijacking is involved.
    b. **Mitigation Limitations**: HTTPS alone cannot prevent DNS-based attacks; users in high-risk networks may still be vulnerable.
    c. **Associated CVEs**: CVE-2019-3569 (relates to TLS and SSL vulnerabilities in web applications).
    d. **Threat Actors**: Cybersecurity students or attackers who have access to tools like Wireshark, Bettercap, or MITM frameworks might experiment with MITM attacks.

3. **Information Disclosure from Misconfigured Headers or APIs**
    a. **Description**: In certain edge cases, misconfigured HTTP headers or unfiltered API responses could inadvertently expose sensitive information.
    b. **Mitigation Limitations**: While Content Security Policies (CSP) and response filtering are implemented, complex user-generated content or unexpected data exposure from third-party integrations may still result in information leaks.
    c. **Associated CVEs**: CVE-2019-15511 (exposing sensitive information due to improper configurations).
    d. **Threat Actors**: Students majoring in cybersecurity or interns seeking vulnerabilities could probe the site for these types of misconfigurations, hoping to find exposed data.

4. **Denial of Service (DoS) from Distributed Attacks**
    a. **Description**: Even with rate limiting and load balancing in place, a large-scale, distributed DoS attack could overwhelm the server's resources.
    b. **Mitigation Limitations**: Full mitigation against DDoS attacks requires specialized infrastructure (e.g., content delivery networks or cloud-based DDoS protection), which may be outside the project's scope.
    c. **Associated CVEs**: CVE-2018-1002105 (DoS vulnerability in Kubernetes, relevant for distributed systems).
    d. **Threat Actors**: Potential threat actors include malicious hackers and curious students running automated tools or botnets like LOIC, Metasploit modules, or Kali Linux scripts.

5. **Privilege Escalation via Undiscovered Vulnerabilities**
    a. **Description**: Privilege escalation may still be possible through zero-day vulnerabilities or unknown application flaws, allowing attackers to gain unauthorized access to restricted areas.
    b. **Mitigation Limitations**: Regular vulnerability scans help, but zero-day attacks exploit unknown flaws that remain vulnerable until patches are available.
    c. **Associated CVEs**: CVE-2021-3156 (a privilege escalation vulnerability in common services).
    d. **Threat Actors**: Students studying penetration testing or ethical hacking may attempt privilege escalation using tools like Metasploit, with the intent to explore or test the system.

6. **Insider Threats**
    a. **Description**: Insider threats from authorized users (e.g., other students or admins) may still exist. This could involve misusing granted privileges to access sensitive information.
    b. **Mitigation Limitations**: While RBAC is enforced, it cannot entirely eliminate the risk of an authorized user abusing their permissions.
    c. **Associated CVEs**: CVE-2019-18935 (arbitrary code execution vulnerabilities related to insider threats).
    d. **Threat Actors**: Authorized users with malicious intent, or curious students who experiment with admin privileges, pose a risk if they attempt to test limits within the system.

7. **Web Application Vulnerabilities**
    a. **Description**: Vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), or Cross-Site Request Forgery (CSRF) may persist in niche areas, despite general input validation.
    b. **Mitigation Limitations**: Full prevention requires continuous testing and patching. Advanced XSS or SQL injection vectors could bypass filters.
    c. **Associated CVEs**: CVE-2017-5638 (Apache Struts vulnerability, related to injection attacks).

    d. **Threat Actors**: Cybersecurity students, particularly those interested in ethical hacking, may leverage tools like SQLmap or Burp Suite to identify and exploit weaknesses in the system.

6.2 Potential Threat Actors

- **University of Wollongong (UOW) Cybersecurity Students**: Students may attempt to test the system using techniques learned in class, such as ping sweeps, vulnerability scanning, and Metasploit testing. While often benign, these actions can still result in accidental disruptions.

- **Ethical Hackers or Penetration Testers**: With access to specialized tools and techniques, these individuals might explore the system as a part of vulnerability assessments, which may expose security flaws.

- **Malicious Hackers**: While less likely in a local application, there remains the risk that someone with malicious intent may actively seek vulnerabilities to exploit.

- **Insiders with Elevated Privileges**: Any users with special access rights, such as admins, might pose risks if their credentials are compromised or they act against policy.

Conclusion

The residual risks identified above are inherent in web applications and are influenced by factors such as potential zero-day vulnerabilities, complex attack scenarios, and resource constraints in full-scale mitigation. The booking system's security posture would benefit from regular vulnerability assessments, periodic security audits, and continuous monitoring, especially considering its use in an academic setting where cybersecurity students may be interested in exploring the system's defenses.

Ultimately, while these residual risks can be minimized, they highlight the importance of continuous security diligence to protect against evolving threats.

7. Conclusion

The threat model for the booking system has highlighted various potential vulnerabilities across key system components, identified security goals, and proposed mitigations to address threats. However, as with any web application, some residual risks remain due to limitations in current technology, inherent complexities, and the evolving nature of cybersecurity threats.

Summary of Key Points

- **STRIDE Analysis**: The STRIDE approach was applied to identify key threats across system components, revealing areas that may be susceptible to spoofing, tampering, information disclosure, and denial of service, among others.
- **Mitigations and Residual Risks**: Effective mitigations, such as JWT-based authentication, rate limiting, input validation, and RBAC, have been proposed for each identified threat. Residual risks, including advanced attacks like zero-day exploits and social engineering, remain but are mitigated as much as feasible within the project's current scope.
- **Potential Threat Actors**: University of Wollongong (UOW) cybersecurity students, ethical hackers, and insiders with access privileges may be interested in testing the system's security, which highlights the need for robust monitoring and logging.

Recommended Next Steps

1. **Implement Regular Security Audits**: Schedule periodic audits, both automated and manual, to identify potential vulnerabilities, especially new ones introduced through updates or configuration changes.
2. **Conduct Penetration Testing**: Engage in penetration testing activities, ideally by third-party security experts, to validate the mitigations in place and identify any overlooked vulnerabilities.
3. **Apply Continuous Monitoring and Logging**: Set up monitoring and alerting for any unusual activity. Logging can be essential for detecting breaches and performing incident analysis should an attack occur.
4. **Educate Users and Admins**: Provide basic security training for all users, especially those with administrative privileges, to help prevent social engineering attacks and insider threats.
5. **Update and Patch Regularly**: Keep all libraries, frameworks, and dependencies up to date to minimize exposure to known vulnerabilities (CVEs).

Final Remarks

This threat model provides a comprehensive starting point for securing the booking system. However, security is an ongoing process that requires continuous improvement and vigilance. By following the recommended steps and maintaining an adaptive security posture, the system can be safeguarded against current and emerging threats, ensuring a safe experience for all users.

- End -