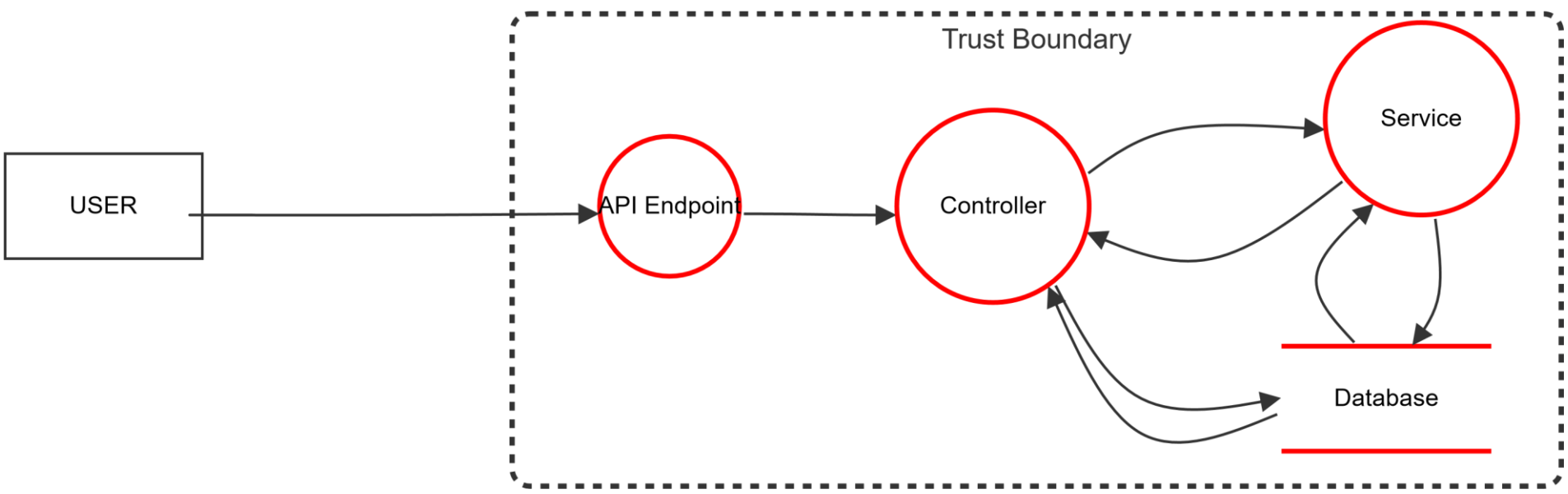
STRIDE THREAT ANALYSIS ON A SELECTED SYSTEM

Executive Summary

Threat analysis on a simple messaging application

Total Threats	8
Total Mitigated	2
Not Mitigated	6
Open / High Priority	1
Open / Medium Priority	5



USER (Actor)

Threats	STRIDE TYPE	Description	Mitigation
Brute Force Attack	Spoofing	Weak passwords are susceptible to brute force attacks and can be easily guessed or cracked.	Enforce strong password policies that require a mix of uppercase, lowercase, numbers, and special characters. Implement password

			expiration and history policies to prevent reuse of old passwords.
Comprised User Data	Spoofing	An attacker can gain access to user login credentials	Use multifactor authentication even if an attacker got access users credentials there is another layer of security to prevent spoofing

API Endpoints

Threat	STRIDE Type	Description	Mitigation
The service layer is overwhelmed by excessive requests, leading to service disruption.	Denial of service	An attacker floods the service with requests to post messages, causing the application to become unresponsive.	Implement rate limiting and request throttling. Use circuit breakers to gracefully handle failures. Monitor and scale services to handle high loads.
An attacker alters request parameters to perform unauthorized actions.	Tampering	An attacker modifies the request parameters to change the content of a message or post under another user's identity.	Validate and sanitize all incoming data. Use secure coding practices to prevent injection attacks

Database (Storage)

Threats	STRIDE Type	Description	Mitigation
An attacker alters request parameters to perform unauthorized actions.	Tampering	An attacker modifies the request parameters to change the content of a message or post under another user's identity.	Validate and sanitize all incoming data. Use secure coding practices to prevent injection attacks (e.g., SQL Injection, Cross-Site Scripting).

Controllers and Services (Processes)

Threat	STRIDE Type	Description	Mitigation
Sensitive information leakage	Information Disclosure	Sensitive information is exposed through the controller. An attacker receives detailed error messages revealing system internals or gains access to	implement proper error handling to avoid leaking information. Enforce strict access controls on the data returned by

		unauthorized data via the controller.	controllers. Ensure that only necessary data is exposed in the API response.
An attacker gains elevated privileges through the controller.	Elevated Privilege	A user exploits a flaw in the controller to perform actions reserved for admin users, such as deleting messages.	Enforce strict role-based access control (RBAC) in the controller. Validate user roles and permissions on every request. Regularly review and update access control rules.
User denies performing transaction	Repudiation	A user denies performing actions that interact with the repository. A user denies having deleted a message, and there's no evidence to prove otherwise.	Implement comprehensive logging of all interactions with the repository. Use versioning or audit tables to track changes to data.

Threats Mitigated by Output Encoding

a. Cross-Site Scripting (XSS)

Threat: XSS occurs when an attacker injects malicious scripts into web pages viewed by other users. These scripts can execute in the context of the victim's browser, potentially stealing cookies, session tokens, or other sensitive information.

Solution with Output Encoding: By properly encoding output, especially when displaying user-generated content, the application ensures that any HTML or JavaScript code provided by an attacker is treated as plain text, not executable code. For example, <script> tags are encoded as <script>, preventing execution.

b. HTML Injection

Threat: Similar to XSS, HTML Injection occurs when an attacker injects arbitrary HTML into the page. This could lead to content injection, phishing, or altering the appearance of the site.

Solution with Output Encoding: Proper encoding ensures that HTML elements are displayed as text rather than rendered as part of the document structure, preventing unintended modifications to the page layout or content.

Threats Mitigated by Input Validation