SECURE CODING: THREAT MODELING

P04:TRADEUP

<TEAM MEMBER NAMES & IDS>

STUDENT ID	NAME
26100216	MUHAMMAD RAYYAN KHAN
26100200	MUHAMMAD AHMAD
26100204	MUHAMMAD SHAHMIR SHER QAZI
26100355	MUHAMMAD UMAR ZUBAIR
25100137	MOHAMMED RAIYAAN JUNAID HAMID

TABLE OF CONTENTS

1. Introduction

The project is a **stock trading simulation platform** that enables hands-on learning for people who want to gain trading experience without the risk of losing money. The platform is intended to provide experiential learning value to users, to supplement their stock trading learning journeys.

Our platform's objectives are to provide a safe and risk-free trading environment that pulls real-time stock market data to simulate the real thing – meaning our environment will capture and simulate the market's movement as it moves in real-time. Furthermore, we aim to provide valuable learning experiences to users with our platform that engage and empower them to improve their stock trading abilities and confidence.

Proposed Feature Set
- Buying/Selling Stocks
- Gamified Leaderboard and Topic-Specific Channels
- AI Insights and Chatbot
- AI News Sentiment Analysis
- Candlestick Charts View
- MarketWatch: to track favorited stocks
- Educational content
- Personalised Notification Systems

2. Instructions

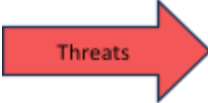Do the following for each sprint:

- Use **STRIDE** framework to perform threat modeling for each use case of the sprint. Focus only on the use cases that you plan to develop in the sprint.
- Analyze the functionality of each use case to assess the possibility of threats from each aspect of the STRIDE framework, i.e., spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege.
- For each threat, identify the mitigation strategies (i.e., the controls that you will build in your application) to address the threat under consideration.
- Examples are given in this document and slides. Make sure that you strictly follow the pattern given in the examples.
- This document will evolve overtime as threat modeling for use cases of the future sprints are added.
- This document must be submitted along with each of the project deliverables in the future. It must also be kept updated.
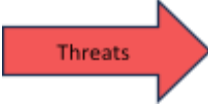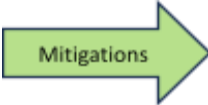
3. Threat Modeling: Sprint-1

<Perform STRIDE threat modeling for each use case. You must describe briefly the threat and corresponding mitigation strategies. The following three examples provide a baseline to start with. Make sure that you read and follow the instructions given above.>
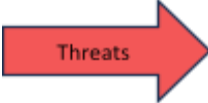
Use Case Name	Spoofing	Tampering	Repudiation	Information disclosure	Denial of service	Elevation of Privilege
Update email/password Threats	An attacker may attempt to log in as the user and update the email/password without authorization.	An attacker may intercept or modify the update request to change email/password to an attacker-controlled	A user may deny that they changed their email/password.	Email update API leaking sensitive user info, or password reset endpoints revealing account existence.	Attackers may flood the email/password update endpoint, preventing legitimate updates.	User may exploit insecure APIs to update another user's email/password.
Mitigations	Use secure session cookies with HttpOnly, Secure, SameSite flags. Implement rate limiting for login attempts.	Use HTTPS for all communication.	Maintain audit logs for account changes.	Avoid exposing email addresses in responses. Mask sensitive data in logs.	Apply rate limiting per user. Use CAPTCHA when abnormal behavior is detected.	Enforce role-based access control (RBAC).

Add profile picture	Attacker uploads an image as another user's profile (ID manipulation).	Attacker uploads malicious files. Attacker replaces another user's picture.	User denies uploading a malicious or inappropriate picture.	Uploaded image path may reveal internal file structure. Image metadata may leak GPS/location.	Attacker uploads extremely large files repeatedly.	Attacker may use file upload to execute code or escalate privileges.
	Validate user session ID on server before updating profile.	Validate MIME type and use server-side file type detection (not just extension). Ensure user ownership check on upload.	Keep upload logs (timestamp, user ID, filename).	Strip metadata (EXIF removal). Store images in private bucket or server folder with signed URLs. Do not expose internal storage paths.	Apply maximum upload limits (e.g., 2MB). Use rate limiting on upload endpoints.	Accept only image formats (JPG/PNG/WebP).

PnL Data Visibility to User						
Threats	Attacker pretends to be the user to view PnL.	Attacker sends manipulated requests to view another user's PnL.	User disputes viewing or modifying PnL data.	API might leak sensitive trade data.	Attacker might spam the PnL endpoint with heavy queries.	A normal user tries to access admin-level PnL analytics or aggregated data.
Mitigations	Secure cookies, session timeouts.	Enforce server-side access control: a user can only view their own PnL.\n\nEnsure backend recalculates PnL instead of trusting client-sent numbers.	Log PnL query actions (user ID, timestamp).	Encrypt PnL info at rest and in transit.	Rate limiting, caching	Ensure authorization checks on every PnL API endpoint.

Top-Up Wallet with Virtual Currency Threats	Attacker pretends to be another user to load credits into their wallet.	Client-side requests can be modified to submit higher values, bypass limits, or inject malformed payloads. If you trust the UI, you lose.	User denies topping up or disputes the amount..	Wallet API leaks wallet balance or transaction details of other users.	Attacker spams the top-up endpoint, blocking legitimate transactions.	User tries to exploit admin APIs to modify their wallet balance.
Mitigations	Server determines wallet ownership using session ID only.	Reject any client-provided monetary values that violate server-defined rules. Perform all amount validation server-side; ignore client-side constraints.	Maintain secure audit logs of wallet transactions. Store timestamp, amount, payment method, and user ID.	Encrypt sensitive data	Enforce rate limiting	Enforce role-based access control (RBAC). Ensure API is secured from loopholes or glitches.

Ai for Stock simulation	An attacker impersonates the external financial data provider and sends fake market data, causing corrupted predictions.	An attacker modifies the stored ML model file to influence predictions.	A malicious internal user denies triggering a model retraining job that produced incorrect predictions.	Unauthorized users access prediction API logs containing sensitive model parameters or financial data.	Attackers flood the prediction API with high-volume requests, overwhelming resources and stopping normal operations.	A compromised user account gains admin-level access and can alter model configurations.
Threats →						
Mitigations →	Use signed API requests to verify the identity of the data provider before ingesting any data.	Store model artifacts with cryptographic integrity checks and validate signatures before loading the model.	Enable immutable audit logs with timestamps and user IDs.	Apply role-based access control (RBAC) and encrypt all logs both in transit and at rest.	Implement rate limiting on the endpoint to block abusive calls.	Use least-privilege IAM roles and enforce multi-factor authentication (MFA) for all privileged accounts.

View Stock News	Attacker spoofs the external news provider and sends fake/malicious news to the backend.	Attacker intercepts and alters the news data in transit between backend and external API	External API provider or internal systems could deny sending or modifying news results.	News API could return data containing sensitive or internal information not meant for users.	Attacker spams the news endpoint, causing API rate limits or backend resource exhaustion.	Attacker manipulates the backend request parameters to access data beyond allowed scope
	Store API keys securely on the server-side. never expose to frontend.	Only use HTTPS for external API communication.	Maintain audit logs of all news API calls	Validate returned data before sending to frontend.	Implement rate limiting on news endpoints.	Validate and sanitize all backend parameters before sending to external API.

4. Threat Modeling: Sprint-2

5. Threat Modeling: Sprint-3

<to be done in the future before Sprint-3 development >

6. Threat Modeling: Sprint-4

7. Common Threats and Mitigations Strategies/Controls

<Here are common threats and mitigation strategies (controls that you can build to address different threats). This is not an exhaustive list but will be useful when you analyze system use cases for threat modeling. Use this just as a guideline, you are NOT required to implement each control in your system, i.e., implement only those which are necessary and relevant for your system.>

Authentication & Authorization Threats

Threat	Description	Mitigation Strategies/Controls
Weak Passwords and Credential Stuffing	Attackers reuse or guess weak passwords. Credential stuffing happens when attackers use leaked username-password pairs from one service to gain unauthorized access to another where users have reused credentials.	- Enforce strong password policy - Implement Multifactor authentication (MFA) - Limit login attempts - Use CAPTCHA - Enable account lockout after a specific number of failed attempts.
Session Hijacking	Stealing or reusing session tokens. **Session hijacking** (aka *session takeover*) occurs when an attacker steals or predicts a user's active session identifier (session ID, cookie etc.) and uses it to take over that user's authenticated session—effectively impersonating them without knowing their password.	- Use HTTPS to avoid sniffing/eavesdropping, - Secure cookies - HttpOnly (Makes cookie inaccessible to JavaScript), - Secure (cookie is sent only over HTTPS), - SameSite (*strict*: cookie is only sent if the request originates from the same site) - Regenerate session IDs after login, - Short session lifetimes (session expiry). - Implement MFA.
Token Theft (JWT / OAuth)	Token stolen from insecure storage or during transmission. An attacker steals or obtains authentication tokens used to prove a user's identity or access rights, allowing them to impersonate that user or access protected resources without needing their password.	- Store tokens securely (Keychain/Keystore), - Use short-lived tokens, - Rotate refresh tokens, - Sign tokens.
Privilege Escalation	Regular user gains admin-level access.	Implement strict RBAC (Role-Based Access Control)

		- Implement ABAC (Attribute-Based Access Control) - Perform server-side authorization checks - Validate user roles in backend logic
Broken Access Control	User accesses data or functionality they shouldn't. Broken access control usually occurs due to mistakes like unprotected endpoints, excessive trust in client-side checks, insecure direct object reference (IDOR).	- Enforce server-side authorization checks, - Never rely solely on client-side validation, - Use secure API gateways.

Input Validation & Injection Threats

Threat	Description	Mitigation Strategies/Controls
SQL Injection	Attacker manipulates database queries via unvalidated user input.	- Use parameterized queries, - Perform input validation, - Enforce least privilege on database accounts.
XSS (Cross-Site Scripting)	Attacker injects malicious scripts into web pages.	- Sanitize input - Encode output - Use Content Security Policy (CSP).
Command Injection	User input triggers system commands.	- Validate and sanitize input, - Use safe APIs (avoid exec()), - Escape shell metacharacters.
Insecure Deserialization	Attacker alters serialized objects to execute code.	- Validate input types, - Sign or encrypt serialized objects, - Avoid deserialization of untrusted data.
Unvalidated Redirects/Forwards	Attacker redirects users to malicious sites.	- Validate redirect targets, - Use whitelists,

Threat	Description	Mitigation Strategies/Controls
		● Avoid user-controlled URLs.
XML External Entity (XXE)	An XXE attack happens when an XML parser processes malicious XML that defines external entities, which can then be used to read local files, perform network requests, or crash the system. It's a way for attackers to abuse how XML parsers handle external resources.	● Validate and sanitize XML input ● Use less risky data formats, if possible, such as JSON or other formats that don't have external entity features.
Input Overflow / Buffer Overflow	Input exceeds expected length, causing memory corruption.	● Enforce strict length checks and bounds validation.

Data Protection & Cryptographic Threats

Threat	Description	Mitigation Strategies/Controls
Data Leakage	Sensitive data exposed in logs, caches, or storage.	● Mask logs, ● Disable caching for sensitive info, ● Secure storage (Keychain/Keystore), ● Data minimization.
Insecure Data Storage	Sensitive data stored or transmitted in plaintext (unencrypted).	● Encrypt data at rest (AES-256), ● Use OS-provided secure storage, ● Avoid storing plaintext credentials.
Hardcoded Secrets / API Keys	API keys or credentials hardcoded in code or configuration files.	● Use environment variables, ● Secure vaults (e.g., AWS Secrets Manager), ● Avoid storing secrets in source code.
Man-in-the-Middle (MITM)	Attacker intercepts communication.	● Enforce HTTPS, ● Verify SSL certificates.

Application Logic Threats

Threat	Description	Mitigation Strategies/Controls
Cross-Site Request Forgery (CSRF)	Attacker tricks user into executing unwanted actions.	- Use anti-CSRF tokens, - SameSite cookies, - Verify request origins.
Business Logic Abuse	Exploiting flaws in process flow (e.g., skipping payment).	- Validate workflow on server-side, - Enforce transaction integrity checks.
Server-Side Request Forgery (SSRF)	An attacker tricks a server into making HTTP requests to internal or external resources that the attacker normally wouldn't be able to access.	- Validate and whitelist URLs - Block internal/private IP ranges - Sanitize and canonicalize input

Miscellaneous Threats

Threat	Description	Mitigation Strategies/Controls
Clipboard / Screenshot Leakage	Sensitive info copied or visible in screenshots.	- Avoid copying sensitive data, - Disable screenshots for sensitive screens.
Insecure Third-Party SDKs	Vulnerabilities or tracking from libraries.	- Vet SDKs for security, - Limit permissions, - Use trusted sources. - Don't use outdated libraries
Insecure CORS Configuration	Overly permissive cross-origin access.	- Restrict origins, - Disallow *, - Validate allowed domains.

Sensitive Data in URL	Data exposed in logs or browser history.	● Use POST for sensitive requests, avoid query parameters for secrets.
Insecure File Uploads	Uploads allow malicious code execution.	● Validate file types, scan for malware, store outside web root.
API Security	Excessive or automated requests.	● Implement authentication and authorization for all endpoints. ● Validate and sanitize all input data (including JSON, XML). ● Implement rate limiting and throttling to prevent abuse. ● Avoid exposing internal object references (IDs, filenames). ● Use API gateways or WAFs for additional protection. ● Ensure proper CORS configuration — only allow trusted domains. ● API keys. ●
Improper Exception Handling	Crashes or unhandled exceptions reveal internals.	● Catch and handle exceptions gracefully ●
Sensitive data in logs	Logging passwords, tokens	● Mask or omit sensitive data in logs.
Information Disclosure	Revealing system details in error messages.	● Return generic errors to users ● log detailed errors securely.

8. STRIDE Framework and OWASP Top 10

There is **NO strict mapping** between STRIDE and OWASP. However, If you use STRIDE thoroughly to identify threat types while keeping OWASP top 10 in mind, they provide complementary coverage of threats and vulnerabilities. The following table shows an approximate mapping.

STRIDE CATEGORY	OWASP Top 10
SPOOFING Impersonating a user, system, or service	A07 – Identification & Authentication Failures A01 – Broken Access Control
TAMPERING Unauthorized modification of data or requests	A01 – Broken Access Control A03 – Injection (SQLi, XSS, etc.) A08 – Software & Data Integrity Failures
REPUDIATION Lack of proper logging or audit trails	A09 – Security Logging & Monitoring Failures A04 – Insecure Design
INFORMATION DISCLOSURE Exposure of sensitive information	A02 – Cryptographic Failures A04 – Insecure Design A05 – Security Misconfiguration
DENIAL OF SERVICE Degrading or blocking system availability	A06 – Vulnerable & Outdated Components A10 – Server-Side Request Forgery (SSRF)
ELEVATION OF PRIVILEGE Gaining privileges beyond authorization	A01 – Broken Access Control A04 – Insecure Design

9. Security Engineer

< Each team must designate one member as the **Security Engineer**. While the entire team is responsible for implementation of the project's security features, the Security Engineer will take the lead in overseeing and ensuring the overall security of the project. **>**

Name of the Security Engineer	Umar Zubair

10. Use of Generative AI

<Mention here how generative AI was used in preparation of this artifact.>

11. Who Did What?

Name of the Team Member	Use Cases
Muhammad Rayyan Khan	Edit email/pw/name, upload profile picture, Stock news page
Muhammad Ahmad	Stock news page
Muhammad Umar Zubair	Ai stock simulation
Mohammad Raiyaan Junaid Hamid	User PnL, Virtual Currency Topup
Muhammad Shamir Sher Qazi	Ai stock simulation

12. Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Section Title	Reviewer Name(s)
Threat modeling sprint-1	Muhammad Rayyan Khan