Assignment 3: Secure Cloud Application Architecture & Automated Threat Modeling Plan

Aastik Bansal [23114002], Abdullah Azeem [23114003]

February 13, 2026

Contents

1 Introduction

Modern cloud applications face evolving security threats due to their distributed nature. This project aims to design a robust architecture that not only withstands attacks but also proactively identifies and mitigates risks through automation. The system will simulate a real-world environment with a client, API gateway, application servers, and a secure data storage, monitored by a central security module.

2 System Architecture

The proposed architecture follows a defense-in-depth approach, ensuring security at multiple layers.

2.1 Components

- **Client**: A web-based interface for user interaction.
- **API Gateway / Reverse Proxy**: The entry point for all external traffic, handling SSL termination, rate limiting, and initial request validation.
- **Application Server(s)**: Backend services processing business logic, running in a private subnet.
- **Data Storage Service**: A secure database (e.g., PostgreSQL) for persistent data, accessible only by the application servers.
- **Security Monitoring & Logging Module**: A centralized service aggregating logs, detecting anomalies, and triggering automated mitigation.

2.2 Trust Boundaries

- **External vs. DMZ**: The boundary between the public internet and the API Gateway.
- **DMZ vs. Private Network**: The boundary between the API Gateway and the internal Application Servers.
- **Application vs. Data**: The restricted access channel between the App Server and the Database.

3 Authentication and Authorization

To ensure secure access control, the system implements a robust identity management strategy.

3.1 Authentication (AuthN)

We utilize **JSON Web Tokens (JWT)** for stateless authentication.

- **Token Issuance**: Upon successful login, the server issues an Access Token (short-lived, e.g., 15 minutes) and a Refresh Token (long-lived, e.g., 7 days).
- **Storage**: Access tokens are stored in memory (or short-lived cookies), while refresh tokens are stored in secure, HttpOnly cookies to prevent XSS attacks.
- **Validation**: The API Gateway validates the JWT signature (HMAC-SHA256 or RSA) on every request before forwarding it to the application server.

3.2 Authorization (AuthZ)

Role-Based Access Control (RBAC) is enforced to restrict access based on user roles.

- **Admin**: Full access to system configurations, user management, and logs.
- **User**: Access to own profile and standard application features.
- **Auditor**: Read-only access to security logs and threat reports.

Policy enforcement points (PEPs) are located at the API Gateway for course-grained control and at the Application Layer for fine-grained control.

4 Threat Modeling (STRIDE)

We apply the STRIDE methodology to identify potential threats across the system components.

Threat Type	Component	Description
Spoofing	Client / API	Attacker impersonating a legitimate user via stolen tokens.
Tampering	Data Storage	SQL injection or modifying data in transit.
Repudiation	App Server	User denying an action due to lack of sufficient logging.
Information Disclosure	API	Leaking sensitive data (PII) in error messages or logs.
Denial of Service	API Gateway	Overwhelming the system with requests (DDoS).
Elevation of Privilege	App Server	Regular user gaining admin access via RBAC flaw.

Table 1: STRIDE Threat Analysis

5 Automated Risk Mitigation Strategies

The system enables proactive defense mechanisms triggered by real-time events.

5.1 Rate Limiting & Throttling

Implemented at the API Gateway using a *Leaky Bucket* or *Token Bucket* algorithm.

- **Global Limit**: Max 1000 requests/minute to protect infrastructure.
- **Per-IP Limit**: Max 60 requests/minute per client IP.

5.2 Automated Blocking (Fail2Ban Style)

The Security Module monitors logs for repeated failures.

- **Condition**: If an IP addresses generates > 5 failed login attempts within 1 minute.
- **Action**: The IP is added to a temporary blacklist for 15 minutes.
- **Account Lockout**: After 5 consecutive failed password attempts, the account is locked for 30 minutes, requiring email verification to unlock.

5.3 Circuit Breakers

To prevent cascading failures during high load or partial outages, circuit breakers are used in inter-service communication. If a service (e.g., Database) fails repeatedly, the circuit opens, returning a strict error immediately instead of waiting for timeouts.

5.4 Input Validation & Sanitization

Strict whitelist-based validation is enforced at the API Gateway to neutralize Injection attacks (SQLi, XSS) before they reach the application logic.

6 3-Week Simulation Plan

6.1 Week 1: Foundation & Architecture

Objective: Establish the core infrastructure and secure communication channels.

- **Days 1-2**: Set up the simulated environment (Docker containers for Client, Gateway, App Server, DB).
- **Days 3-4**: Implement JWT-based Authentication and RBAC mechanisms.
- **Days 5-7**: Configure the API Gateway with SSL/TLS and basic request validation. Establish secure logging pipelines.

6.2 Week 2: Threat Modeling & Attack Simulation

Objective: Identify vulnerabilities and simulate attacks to test defenses.

- **Days 1-3**: Run automated threat modeling tools (e.g., OWASP ZAP, Python scripts) against the architecture.
- **Days 4-7**: Develop and run attack simulation scripts:
  - *Brute-force attack* on the login endpoint.
  - *SQL Injection* attempts on data retrieval endpoints.
  - *DoS Simulation* using high-volume request scripts.

6.3 Week 3: Automated Mitigation & Resilience Evaluation

Objective: Implement automated defenses and measure system recovery.

- **Days 1-3**: Implement the Security Module logic for IP blocking and account lockout.
- **Days 4-5**: Run full-scale resilience tests. Measure:
  - *Time to Detect*: How fast the logging module flags an anomaly.
  - *Time to Mitigate*: Latency between detection and active blocking.
- **Days 6-7**: Compile logs, generate evidence screenshots, and finalize the report.

7 Conclusion

This plan ensures a structured approach to building a secure cloud architecture. By simulating real-world threats and implementing robust AuthN/AuthZ and automated mitigation, we demonstrate the system's resilience and adherence to security best practices.