Enhanced Formal Review

Software Requirements Specification (SRS)
Intelligent Cyber Threat Intelligence System

Review Group: Excellence Review Board
Version: 1.1

November 29, 2024

Contents

1 Precondition Document

Group Members:

- Dr. Jane Smith
- John Doe
- Maria Gonzales
- Rahul Patel
- Emily Chen

Application Description: The project under review is a Cyber Threat Intelligence System designed for enterprise-grade threat detection and response. Its key features include:

- Real-time ingestion of threat data.
- Enrichment of data with MITRE ATT&CK classifications.
- Actionable dashboards for security teams.
- Predictive analytics for threat forecasting.
- Integration with SIEM/SOAR platforms.

Planned Testing Activities:

- Unit testing using the AAA pattern and parameterized tests.
- Integration testing of APIs and database interactions.
- Stress and performance testing for ingestion pipelines.
- End-to-end UI testing of role-specific dashboards.
- Security testing to ensure compliance with GDPR and ISO 27001.
- Regression testing for iterative releases.

2 Enhanced Formal Review

2.1 Role Assignment for Review

Name	Role	Responsibilities
Dr. Jane Smith	Lead Reviewer	Review architecture, testing strategy, and SRS alignment.
John Doe	Compliance Specialist	Evaluate legal and regulatory compliance.
Maria Gonzales	Usability Expert	Assess dashboards and end-user workflows.
Rahul Patel	Security Analyst	Critique security mechanisms and risk mitigation.
Emily Chen	Performance Analyst	Analyze performance benchmarks and stress testing.

2.2 Detailed Observations and Critique

2.2.1 Introduction

Key Issues:

- Ambiguity in project goals. No clear success metrics provided.
- Lack of differentiation between intended user groups (e.g., SOC analysts, executives).

Improvements:

- Define measurable outcomes, such as: "Reduce SOC analyst response time by 30
- Clearly specify use cases for each user group.

2.2.2 Overall Description

Key Issues:

- Over-reliance on Azure-specific services, ignoring multi-cloud or hybrid options.
- Unclear preconditions and dependencies for threat ingestion pipelines.

Improvements:

- Incorporate alternative cloud strategies for vendor neutrality.
- Document fallback mechanisms for precondition failures.

2.2.3 Specific Requirements

Functional Requirements:

- FR-1 lacks details on API versioning and error handling mechanisms.
- FR-4 does not specify how RBAC will be enforced for different roles.

Non-Functional Requirements:

- NFR-1 does not specify system recovery time after failure.
- NFR-3 encryption requirements omit specific compliance benchmarks.

Improvements:

- Enhance API documentation with explicit error codes, rate limits, and versioning strategy.
- Specify encryption standards (e.g., FIPS 140-2) and recovery time objectives (RTO).

2.2.4 Testing and Risk Management

Key Issues:

- Testing strategies lack depth; no mention of parameterized tests or stress testing design.
- Risk table ignores critical risks, such as vendor lock-in and cascading failures.

Improvements:

- Include examples of test cases, such as parameterized tests for API inputs and outputs.
- Expand risk assessment to include contingency plans for vendor lock-in.

2.2.5 System Architecture

Key Issues:

- Inadequate focus on security layers in microservices.
- Over-reliance on Azure services without consideration of on-premise deployments.

Improvements:

- Define a layered security architecture for microservices.
- Provide an abstract deployment model adaptable to non-Azure environments.

3 Exam-Related Questions and Insights

3.1 Testing Techniques

- **Boundary Values Technique:** Applied to API rate limits and input validation.
- **Decision Table:** Omitted due to a lack of complex conditional logic in workflows.

3.2 Unit Testing and Patterns

- **AAA Pattern:** Used for testing enrichment services by arranging test data, executing transformations, and asserting outputs.
- **Parameterized Tests:** Employed for API endpoint tests using frameworks like JUnit and TestNG.
- **Test Doubles:** Used mock databases and APIs to isolate components during integration tests.

3.3 Database and API Testing

- **Database Testing:** Validated schema integrity, query performance, and transactional consistency.
- **API Testing:** Designed contract tests for external APIs with Swagger and tested failure scenarios.

3.4 Performance Testing

- **Stress Tests:** Simulated ingestion of 100k IOCs to verify system resilience.
- **Spike Tests:** Tested system response to sudden surges in threat data ingestion.

3.5 Key Answers for Theoretical Questions

- **Testing vs Debugging:** Testing identifies defects; debugging locates and resolves the cause.
- **Regression Testing:** Ensures that new code changes do not break existing functionality.
- **Test Pyramid:** Prioritizes unit tests over integration and UI tests for efficiency.

4 Conclusion

Overall Rating: **3.5/5** While the document demonstrates potential for real-world application, it falls short in clarity, testing depth, and adaptability.