# Final Project – Phase I: Requirements Assignment

**Project Name:** Secure Art Gallery Log

**Team Name:** Group 6

**Team Members:**

- Sarvesh Patil
- Jay Eichmuller

1. Project Overview

Our project is a **secure logging system** for tracking activity inside an art gallery. The system keeps track of:

- Guests and employees entering and leaving.
- Who is currently in each room.

It will be implemented as two programs in C++:

- **logappend** → appends new entries to the log file (e.g., person enters/exits).
- **logread** → reads the log and answers queries about the current gallery state.

Both programs use an **authentication token** (command-line argument) to make sure only authorized users can read or write the log.

The environment:

- Runs on Linux/Unix command line.
- Written in C++ with a focus on secure coding.
- Tested in a classroom/lab environment.

2. Functional Security Requirements

- **Authentication:** Each program (logappend, logread) requires a valid authentication token passed via command-line. Invalid tokens are rejected.
- **Authorization:** Only authorized users can append logs or read logs. Role-based restrictions may be added (for example, log readers cannot modify logs).

- **Data Protection:** Log file entries must not be altered or tampered with. Integrity will be checked using cryptographic hashing or a similar method.
- **Input Validation:** All command-line arguments will be validated. This prevents buffer overflows, injection attacks, and ensures string lengths are within limits.

- **Logging & Monitoring:** Security events such as invalid tokens, malformed input, or unauthorized actions will be logged for debugging and auditing.

3. Non-Functional Security Requirements

- **Performance & Scalability:** System should still work correctly if the log file grows large (many entries).
- **Availability:** The log file should remain accessible even if an invalid entry is attempted. Corruption prevention is important.
- **Usability:** Commands for logappend/logread should be simple and consistent, so security doesn't make the system too hard to use.
- **Compliance:** Since this is a class project, there are no real-world compliance laws, but we will follow **secure C++ practices** (memory safety, avoiding vulnerabilities like buffer overflows).

4. Security Goals and Objectives

- Protect the **confidentiality, integrity, and availability (CIA)** of the log file.
- Prevent tampering of log data.
- Ensure only authenticated/authorized users can access the log.
- Make all actions **auditable** (so invalid attempts are recorded).

5. Compliance and Legal Requirements

Not directly applicable since this is an academic project. However:

- We will follow **NIST Secure Software Development Framework (SSDF)** practices.
- We will reference **OWASP guidelines** for input validation and secure coding.

6. Risk Assessment and Prioritization

**Risk:** Buffer Overflow

- **Description:** User enters input longer than expected, causing memory corruption.
- **Impact:** High
- **Mitigation:** Validate input length, use safe string handling in C++.
- **Priority:** High

**Risk:** Unauthorized Access

- **Description:** Invalid user tries to append/read log.
- **Impact:** High
- **Mitigation:** Require authentication token, reject invalid attempts.
- **Priority:** High

**Risk:** Log Tampering

- **Description:** Attacker modifies log file manually.
- **Impact:** Medium
- **Mitigation:** Integrity check (e.g., hashing) before reading log.
- **Priority:** Medium

**Risk:** Data Loss

- **Description:** Log file gets corrupted or accidentally deleted.
- **Impact:** Medium
- **Mitigation:** Use safe file write operations, backups if possible.
- **Priority:** Medium

**Risk:** Performance Degradation

- **Description:** Very large log slows queries.
- **Impact:** Low
- **Mitigation:** Optimize parsing, handle incremental reads.
- **Priority:** Low

7. Summary of Security Requirements

Functional Requirements

- Authentication token required.
- Authorization enforced for append/read.
- Input validation for all parameters.
- Log integrity maintained.
- Security events logged.

Non-Functional Requirements

- Performance under large logs.
- Availability (no corruption on bad input).
- Usability (simple commands).
- Compliance with secure coding practices.

Risks

- High-priority: Buffer overflow, unauthorized access.
- Medium-priority: Log tampering, data loss.
- Low-priority: Performance degradation.