Contents

Document Status:: Finalised For Initial Submission

CITS3007 Secure Coding - Group Project Phase 1 Report

Oblivionaire Online (OO) - Access Control System (ACS)

Team Name: 3007-Team-Sixteen
GitHub Organisation Name: 3007-Team-Sixteen
Group Number: 16

Group Members

- Stephen Beaver (10423362)
- Kai Fletcher (23808253)
- Prem Patel (23775211)
- Muhammad Qureshi (23976415)
- Kelly Snow (23614821)

Document Format

- Primary format: Markdown (.md)
- Submission format: PDF (.pdf)
- Conversion tool: Pandoc with xelatex
- Fonts: Fira Sans (main text), Fira Code (code blocks)
- Layout: 1-inch margins, colored links, table of contents (depth: 2)

1. Team Communication & Responsibilities

1.1 Communication Strategy

- **Meeting Schedule**:
  - Weekly: Monday 10:00 at UWA (approx. 2 hours, includes Sprint Planning)
  - Standups: Wednesday & Friday (Online via Discord, 5 min check-in)
  - Emergency: Discord/Signal
- **Communication Tools**:
  - Discord: Primary platform with dedicated channels
  - Signal: Secure messaging for urgent communications
  - GitHub Projects: Task tracking and project management
- **Progress Tracking**:

    - Standup updates (Wed/Fri)
    - Weekly reviews (Mon)
    - Burndown charts
    - Peer reviews
    - Documentation updates

1.2 Responsibility Allocation

- **Core Responsibilities**:
    - Technical Lead & Infrastructure (Stephen)
    - Authentication & Security (Kelly)
    - RBAC & Financial Controls (Prem)
    - Session Management & Performance (Muhammad)
    - Testing & Quality Assurance (Kai)
- **Shared Responsibilities**:
    - Code reviews
    - Security awareness
    - Documentation
    - Sprint planning
    - Knowledge sharing

2. Version Control Strategy

2.1 Repository Structure

- Organized directory structure for source code, tests, docs, and tools
- Clear separation of concerns between components

2.2 Branching Strategy

- `main`: Production-ready code
- `develop`: Integration branch
- Feature/bugfix/security branches
- Protected branches with required reviews

2.3 Security Measures

- GitHub commit signing
- Branch protection rules
- Required PR reviews
- Automated security checks

3. Development Tools

3.1 Development Environment

- **Cloud Infrastructure (Linode)**:
    - Development and testing environments
    - Staging and production environments
    - Justification: Provides isolated environments for development, testing, and production with built-in security features and automated deployment capabilities.
- **Infrastructure Automation (Ansible)**:
    - Configuration management
    - Security hardening

  - Deployment automation
  - Justification: Enables consistent, repeatable environment setup and security configurations across all instances, reducing human error and ensuring compliance with security standards.
- **Development Tools**:
  - IDE/Editor: Developer's choice (VS Code, Vim, etc.)
  - Justification: While developers may choose their preferred IDE, all code must pass through the standardized CI/CD pipeline which ensures consistent formatting, security checks, and build processes regardless of development environment.

3.2 Development Standards

- **Coding Standard**: SEI CERT C
  - Justification: Industry-standard secure coding guidelines specifically designed for C programming, providing comprehensive rules for memory safety, input validation, and error handling.
- **Compiler Configuration**:
  - C11 standard compliance
  - Security-focused compiler flags:
    ```
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -std=c11 -pedantic -Wall -Wextra -Werror -Wform
    set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS_DEBUG} -fsanitize=address,undefined")
    ```
  - Static analysis integration
  - Justification: Ensures modern C language features and enables comprehensive static analysis for security vulnerabilities, with specific flags targeting memory safety, format string vulnerabilities, and undefined behavior.
- **Testing Framework**: Unity
  - Unit testing
  - Memory safety verification
  - Coverage analysis
  - Justification: Lightweight, portable testing framework that integrates well with C projects and supports comprehensive security testing.

3.3 Security Tools

- **Static Analysis**:
  - GCC analyzer
  - Valgrind
  - Justification: Provides comprehensive static and dynamic analysis capabilities for identifying security vulnerabilities and memory issues.
- **Version Control (GitHub)**:
  - Branch protection
  - Required reviews
  - Automated security checks
  - Justification: Enables secure collaboration with built-in security features and automated checks to maintain code quality.
- **Cryptographic Libraries**:
  - libsodium for password hashing
  - OpenSSL for general cryptography
  - Justification: Industry-standard cryptographic libraries with proven security track records and active maintenance.

3.4 Collaboration Tools

- **Communication**: Discord
  - Dedicated channels for different aspects
  - Voice channels for pair programming

    - Justification: Provides organized communication channels and real-time collaboration capabilities essential for team coordination.
- **Project Management**: GitHub Projects
    - Task tracking
    - Milestone management
    - Progress visualization
    - Justification: Integrated with version control, enabling seamless tracking of development progress and task management.

4. Key Secure Coding Practices

4.1 Memory Safety

- Safe memory management patterns
- Bounds checking
- Memory leak detection
- Buffer overflow prevention

4.2 Input Validation

- Comprehensive input validation
- Boundary condition testing
- Format string validation
- Input sanitization

4.3 Access Control

- RBAC implementation
- Principle of least privilege
- Permission verification
- Audit logging

4.4 Error Handling

- Consistent error handling patterns
- Proper error reporting
- Secure logging
- Recovery procedures

5. Risk Management & Quality Assurance

5.1 Risk Management

- **Technical Risks**: Memory safety, input validation, access control
- **Operational Risks**: Environment issues, collaboration, time management
- **Mitigation Strategies**: Regular reviews, testing, documentation

5.2 Quality Assurance

- **Code Quality**: Standards compliance, static analysis, reviews
- **Security Testing**: Unit testing, memory safety, input validation
- **Documentation**: Code, security, user, and process documentation

5.3 Implementation Timeline

- **Phase 1 (Completed)**: Planning and setup
- **Phase 2 (3 Weeks)**: Core implementation and testing
- **Phase 3 (Weeks 11-12)**: Demo and presentation

6. Git-Based Tracking

6.1 Progress Tracking

- Regular commit-based tracking
- Standup and weekly reviews
- PR reviews and project updates

6.2 Effort Tracking

- Metrics: Commits, PRs, reviews, documentation
- Triggers: No commits for 3 days, unreviewed PRs > 24h

6.3 Quality Control

- Automated: Pre-commit hooks, static analysis, tests
- Manual: Code reviews, security reviews, documentation

7. Project Rules and Assessment

7.1 Project Rules and Deadlines

- **Weighting**: 30% of final mark
- **Total Marks**: 60
- **Phase 1**: 10 marks (Due: 16 April 2025)
- **Phase 2**: 40 marks (Week 11)
- **Phase 3**: 10 marks (Weeks 11-12)

7.2 Assessment Criteria

- **Content Requirements (5 marks)**:
  - Team communication and responsibilities
  - Version control strategy
  - Development tools
  - Key secure coding practices
  - Risk management and quality assurance
- **Report Quality (5 marks)**:
  - Clarity of presentation
  - Logical organization
  - Strength of justifications
  - Technical accuracy
  - Professional presentation

For detailed implementations, configurations, and extended explanations, please refer to the supplementary document.