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 in-person at UWA (approx. 2 hours, includes Sprint Planning)
  - Standups: Wednesday & Friday 18:00 (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 e.g. thread modelling, secure design
  - Documentation
  - Sprint planning
  - Knowledge sharing

2. Version Control Strategy

2.1 Github Repository Structure

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

2.2 Branching Strategy

The team will use the Agile methodology and focus on developing small deliverables in each sprint (1 week) which build up to the overall project outcomes. The reason for working in short iterations is to obtain regular feedback and ensure that any issues are detected and addressed quickly.

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

Branch Naming Conventions: - `feature/feature-name`: branch for developing new features/functionality - `bugfix/bug-name`: branch for creating a solution to an existing bug - `security/security-change`: branch for adding a specific security practice (if not already included as part of a feature or bugfix)

Commit Message Conventions: Each commit message must be sufficiently verbose to describe exactly what was implemented in the commit. Vagueness should be avoided. An example is shown below. - “Changes to password hashing” -> bad - “Implemented password hashing using MD5 algorithm” -> good

Workflow: When creating a new feature, bugfix, or security change, a separate branch off the main branch must be made following the appropriate naming convention. Development is done in this branch with regular commits. When work on the branch is finished, a pull request (PR) must be made with ideally 1-2 different team members approving the PR. Once the PR is approved, the branch is merged into main.

2.3 Security Measures

- GitHub commit signing
- Branch protection rules
- Required PR reviews
- Automated security checks via CI/CD pipeline

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

Memory safety is relevant to protect sensitive game resources and confidential data from being tampered with by malicious input from players.

How this will be applied in development: - Safe memory management patterns - Bounds checking - Memory leak detection - Buffer overflow prevention

4.2 Input Validation

Input validation is essential to protect against injection attacks due to unsanitized input from players which can result in unexpected behaviour.

How this will be applied in development: - Comprehensive input validation - Boundary condition testing - Format string validation - Input sanitization to detect special characters etc.

4.3 Access Control

Access control and authorisation is relevant to ensure that all actions in the game are undertaken by the appropriate users.

How this will be applied in development: - RBAC implementation - Principle of least privilege - Permission verification - Audit logging

4.4 Error Handling

Error handling is necessary to ensure that unexpected events do not compromise the smooth and reliable operation of the game.

How this will be applied in development: - Consistent error handling patterns - Proper error reporting - Secure logging - Recovery procedures

All of these practices will be enforced by automated testing of all code going through the CI/CD pipeline. Code which does not follow these practices will not pass the pipeline and will require modification.

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/meetings: ensures team alignment and allows issues to be addressed in a timely manner.
  - Automated testing: consistent testing in CI/CD pipeline reduces chances of issues going undetected due to human error.
  - Comprehensive documentation: facilitates transparency and knowledge sharing as all team members can be aware of all parts of the project and take over if resourcing issues arise.

5.2 Quality Assurance

- **Code Quality**: Standards compliance, integrated static analysis, regular 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.