CP Chain: Security Audit Report

DogScan Security Team

July 21st, 2025

Contents

DogScan Security Audit Report

Project	CP Chain
Contract File	`MessageManager.sol`
Audit Date	July 21st, 2025
Report Version	1.0

1. Executive Summary

We conducted a comprehensive security audit of the `MessageManager.sol` contract. This contract serves as a message coordination system for cross-chain operations but exhibits several design and implementation issues. While the contract implements basic access control and reentrancy protection, it suffers from interface inconsistencies, inadequate message validation, and centralization concerns. The most significant issue is the lack of proper message validation in the claim function, which could allow unauthorized message processing under certain conditions.

The audit results revealed **one low-severity issue and several design improvement recommendations** primarily related to message validation logic and code quality.

Overall Risk Rating: Low

We recommend the project team improve message validation logic and enhance interface consistency and governance mechanisms.

2. Audit Scope

The audit scope covers the `MessageManager.sol` contract and its related components:

Contract Information:

- Contract Type: Cross-chain Message Management Contract
- Main Functions: Message sending and claiming, cross-chain communication coordination
- Storage Layer: `MessageManagerStorage.sol`
- Interface Definition: `IMessageManager`

Key Audit Areas:

- Message sending and claiming mechanisms
- Access control patterns
- Cross-chain communication protocols
- Message integrity validation
- Interface compliance

3. Audit Methodology

This audit employed a multi-agent AI security analysis framework specifically designed for smart contract security assessment:

1. Specialized Analysis Modules:

- **Message Integrity Expert**: Reviews message lifecycle and validation logic
- **Access Control Analyst**: Evaluates permission management and centralization risks
- **Interface Compliance Expert**: Checks consistency between interface definitions and implementations
- **Cross-chain Communication Expert**: Analyzes cross-chain message transmission mechanisms
- **Code Quality Expert**: Evaluates code standards and maintainability

2. Comprehensive Analysis:

- Static code analysis focused on message integrity
- Interface compliance verification
- Potential bypass scenario analysis

4. Findings Summary

ID	Title	Severity	Status
L-01	Missing Message Existence Validation in Claim Function	**Low**	**Pending Fix**
I-01	Interface Parameter Order Mismatch	**Information**	**Pending Fix**
I-02	Centralized Access Control Model	**Information**	**Pending Improvement**

ID	Title	Severity	Status
I-03	Spelling Error in State Variable	**Information**	**Pending Fix**
I-04	Limited Input Validation	**Information**	**Pending Improvement**

5. Detailed Findings

[L-01] Missing Message Existence Validation in Claim Function (Low)

Severity: Low

Description The `claimMessage` function only verifies that a message hasn't been claimed before but doesn't validate that the message was actually sent through `sendMessage`. While protected by `onlyTokenBridge` access control, this design allows the authorized caller to claim arbitrary messages that were never legitimately sent.

Technical Details

```
function claimMessage(...) external onlyTokenBridge nonReentrant {
    bytes32 messageHash = keccak256(...);
    require(!cliamMessageStatus[messageHash], "Message not found!");  // Only checks unclaimed
    // Missing: require(sentMessageStatus[messageHash], "Message not sent");
    cliamMessageStatus[messageHash] = true;
}
```

Impact

- Allows claiming of messages that were never sent
- Security vulnerability in validation logic
- Could be exploited for unauthorized operations

Recommendation Add validation to ensure messages were actually sent before allowing claims:

```
function claimMessage(...) external onlyTokenBridge nonReentrant {
    bytes32 messageHash = keccak256(...);
    require(sentMessageStatus[messageHash], "Message not sent");  // Add
        validation
    require(!cliamMessageStatus[messageHash], "Already claimed");
    cliamMessageStatus[messageHash] = true;
}
```

[I-01] Interface Parameter Order Mismatch (Information)

Severity: Information

Description The `claimMessage` function implementation has parameter order (`_value`, `_fee`, `_nonce`) while the interface `IMessageManager` defines it as (`_fee`, `_value`, `_nonce`). This discrepancy could cause integration issues and confusion for developers.

Impact

- Interface and implementation inconsistency
- Could lead to integration errors
- Affects code maintainability

Recommendation Align the function signature with the interface definition to maintain consistency.

[I-02] Centralized Access Control Model (Information)

Severity: Information

Description The contract relies entirely on a single `poolManagerAddress` for access control to critical functions. This creates a single point of failure where compromise or misconfiguration of this address could affect the entire message system.

Impact

- Single point of failure risk
- Lack of governance transparency
- System security depends on single entity

Recommendation Consider implementing multi-signature access control or a more distributed governance model for critical operations.

[I-03] Spelling Error in State Variable (Information)

Severity: Information

Description The mapping `cliamMessageStatus` contains a spelling error (should be `claimMessageStatus`). While this doesn't affect functionality, it could lead to confusion during integration and maintenance.

Impact

- Affects code readability
- Reduces professional standards
- Could cause maintenance difficulties

Recommendation Correct the spelling to `claimMessageStatus` throughout the contract.

[I-04] Limited Input Validation (Information)

Severity: Information

Description The contract performs minimal input validation beyond checking for zero addresses in some functions. Additional validation could prevent edge cases and improve contract robustness.

Recommendation Implement comprehensive input validation for all parameters including range checks and additional address validations.

6. Architecture and Design Assessment

Design Strengths

1. **Separation of Concerns**: Properly separates concerns by delegating message processing logic to PoolManager while maintaining message state
2. **Reentrancy Protection**: Uses OpenZeppelin's upgradeable patterns and reentrancy protection
3. **Modular Design**: Good storage abstraction and interface definitions
4. **Access Control**: Implements basic access control mechanisms

Key Architectural Issues

1. **Centralized Access Control**: Complete reliance on single address for permission management
2. **Incomplete Message Validation**: Lacks comprehensive message validation logic
3. **Interface Inconsistency**: Implementation differs from interface definitions
4. **Code Quality Issues**: Contains spelling errors and naming inconsistencies

Systemic Risk Assessment

This contract serves as a cross-chain message coordination system with the following characteristics:

1. **Centralized Access Control**: Relies on single `poolManagerAddress` for permission control
2. **Message Validation Defects**: Claim functionality lacks complete validation logic
3. **Interface Compliance Issues**: Implementation inconsistent with interface definitions

7. Conclusion

This security audit identified **one low-severity issue and several design improvement recommendations** in the `MessageManager` contract.

Key Findings Summary:

- **Message Validation Defects**: Claim functionality lacks complete message validation logic

- **Interface Inconsistency Issues**: Function implementation differs from interface definitions
- **Centralized Governance Risk**: Access control overly dependent on single address
- **Code Quality Issues**: Contains spelling errors and insufficient validation

Overall Risk Rating: Low

The contract provides essential functionality for cross-chain message coordination but requires refinements in validation logic and interface consistency. While no high-severity vulnerabilities were identified, the lack of comprehensive message validation and centralized access control present moderate risks.

Priority Remediation Recommendations:

1. **Medium Priority**: [L-01] Improve message validation logic
2. **Medium Priority**: [I-01] Fix interface parameter order
3. **Low Priority**: [I-03] Correct spelling errors
4. **Long-term Improvement**: Implement decentralized governance mechanisms

We recommend the project team address message validation and interface consistency issues before production deployment.