CP Chain: Security Audit Report

DogScan Security Team

July 21st, 2025

Contents

DogScan Security Audit Report

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

1. Executive Summary

We conducted a comprehensive security audit of the `PoolManager.sol` contract. This contract implements cross-chain bridging and staking mechanisms with extensive functionality, but suffers from inadequate state management in withdrawal operations. While the contract uses reentrancy protection in most functions, a fundamental flaw in the withdrawal logic could lead to accounting discrepancies. Additionally, the contract demonstrates significant centralization risks with a single relayer address controlling critical operations.

The audit results revealed **one medium-severity issue** primarily related to state inconsistency during ETH withdrawal processes that requires immediate remediation.

Overall Risk Rating: Medium

We recommend the project team immediately fix the state inconsistency issue and consider implementing more decentralized governance mechanisms.

2. Audit Scope

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

Contract Information:

- Contract Type: Cross-chain Bridge and Staking Contract
- Main Functions: Pool management, cross-chain bridging, staking rewards
- Storage Layer: `PoolManagerStorage`
- Interface Definition: `IPoolManager`

Key Audit Areas:

- ETH and ERC20 token withdrawal mechanisms
- Cross-chain bridge finalization functions

- Staking and reward systems
- Access control mechanisms
- State management and accounting integrity

3. Audit Methodology

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

1. Specialized Analysis Modules:

- **State Management Expert**: Reviews state variable update order and consistency
- **Access Control Analyst**: Evaluates permission management and centralization risks
- **Cross-chain Bridge Expert**: Analyzes bridging logic and security
- **Staking Mechanism Expert**: Evaluates staking and reward distribution logic
- **Reentrancy Expert**: Checks reentrancy protection implementation
- **Mathematical Security Expert**: Reviews arithmetic operations and overflow protection

2. Comprehensive Analysis:

- Static code analysis focused on state management patterns
- Access control mechanism verification
- Checks-Effects-Interactions pattern compliance review

4. Findings Summary

ID	Title	Severity	Status
M-01	State Inconsistency in ETH Withdrawal Function	**Medium**	**Pending Fix**
I-01	Excessive Centralization in Access Control	**Information**	**Pending Improvement**
I-02	Missing Token De-support Functionality	**Information**	**Pending Improvement**

ID	Title	Severity	Status
I-03	Checks-Effects-Interactions Pattern Deviation	**Information**	**Pending Improvement**

5. Detailed Findings

[M-01] State Inconsistency in ETH Withdrawal Function (Medium)

Severity: Medium

Description The `withdrawEthFromBridge` function updates the internal balance tracking (`FundingPoolBalance`) before performing the external ETH transfer. If the external call fails, the function returns false but the internal accounting has already been modified, creating a discrepancy between recorded and actual balances.

Technical Details

```
function withdrawEthFromBridge(address payable withdrawAddress, uint256 amount)
    public payable onlyWithdrawManager returns (bool) {
    require(address(this).balance >= amount, "insufficient ETH balance");
    FundingPoolBalance[ETHAddress] -= amount;  // Updates record first
    (bool success, ) = withdrawAddress.call{value: amount}("");  // External call
    if (!success) {
        return false;  // Failure but state already changed
    }
    // ...
}
```

Impact

- Creates discrepancy between recorded and actual balances when transfer fails
- Could lead to fund lockup and broken contract invariants
- Affects other functions that depend on `FundingPoolBalance`

Recommendation Reorder the function logic to perform the external ETH transfer before updating internal state. Only modify `FundingPoolBalance` after confirming the transfer was successful.

```
function withdrawEthFromBridge(address payable withdrawAddress, uint256 amount)
    public payable onlyWithdrawManager returns (bool) {
    require(address(this).balance >= amount, "insufficient ETH balance");

    (bool success, ) = withdrawAddress.call{value: amount}("");
    if (!success) {
        return false;  // Transfer failed, don't modify state
    }

    FundingPoolBalance[ETHAddress] -= amount;  // Update after success
    emit WithdrawToken(ETHAddress, msg.sender, withdrawAddress, amount);
    return true;
}
```

[I-01] Excessive Centralization in Access Control (Information)

Severity: Information

Description The contract relies heavily on a single `relayerAddress` for critical operations including token management, fee settings, pausing, and asset transfers. This creates a single point of failure where compromise of this address could lead to complete system control.

Impact

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

Recommendations

- Implement multi-signature wallet or time-locked governance
- Consider decentralized governance mechanisms (such as DAOs)
- Increase governance transparency

[I-02] Missing Token De-support Functionality (Information)

Severity: Information

Description The `setSupportERC20Token` function can add tokens to the `SupportTokens` array but cannot remove them when setting `isValid` to false, leading to an ever-growing array.

Recommendation Implement array removal logic when disabling token support to maintain consistency between mapping and array.

[I-03] Checks-Effects-Interactions Pattern Deviation (Information)

Severity: Information

Description Some functions perform external calls before state updates, which while currently protected by reentrancy guards, deviates from best practices.

Recommendation Follow the Checks-Effects-Interactions pattern consistently for better security hygiene.

6. Architecture and Design Assessment

Design Strengths

1. **Use of Mature Libraries**: Built on OpenZeppelin's upgradeable patterns
2. **Reentrancy Protection**: Most functions correctly implement reentrancy protection
3. **Modular Design**: Good storage separation and interface definitions
4. **Multi-token Support**: Supports ETH and multiple ERC20 tokens

Key Architectural Issues

1. **Centralized Governance Structure**: Single `relayerAddress` controls critical functions
2. **Inconsistent State Management**: Some functions violate CEI pattern
3. **Array Management Defects**: Missing token removal functionality

Systemic Risk Assessment

This contract adopts a centralized management model with the following characteristics:

1. **Centralized Access Control**: `relayerAddress` has extensive administrative privileges
2. **State Management Issues**: State inconsistency in withdrawal functions could lead to fund accounting errors
3. **Governance Transparency**: Lacks decentralized governance mechanisms

7. Conclusion

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

Key Findings Summary:

- **State Inconsistency Issue**: Logic error in ETH withdrawal function could lead to fund accounting discrepancies
- **Centralized Governance Risk**: Single address controls critical functions
- **Code Quality Issues**: Some implementations deviate from security best practices

Overall Risk Rating: Medium

The contract requires immediate attention to fix the state inconsistency issue to ensure fund security. It is also recommended to improve governance mechanisms and code quality.

Priority Remediation Recommendations:

1. **Immediate Fix**: [M-01] ETH withdrawal function state inconsistency
2. **High Priority**: Implement decentralized governance mechanisms
3. **Medium Priority**: Improve token management functionality and CEI pattern compliance

We recommend the project team fix all identified issues before production deployment.

Disclaimer

This audit report is provided for informational purposes only and does not constitute investment advice. The analysis is based on smart contract source code provided at a specific point in time and does not constitute an endorsement of the project or a guarantee of its security. Smart contracts carry inherent risks, including potential undiscovered vulnerabilities. Users should conduct their own research and exercise caution when interacting with any smart contract. The findings in this report are an assessment of risk based on known attack vectors and best practices.