

Hashlock - Audit Report

 aiaudit.hashlock.com/audit/69671268-7872-4369-9dc0-3993134248ae



Disclaimer: This AI-generated scan may contain inaccuracies and false positives. For professional audits, consult Hashlock security audit experts.

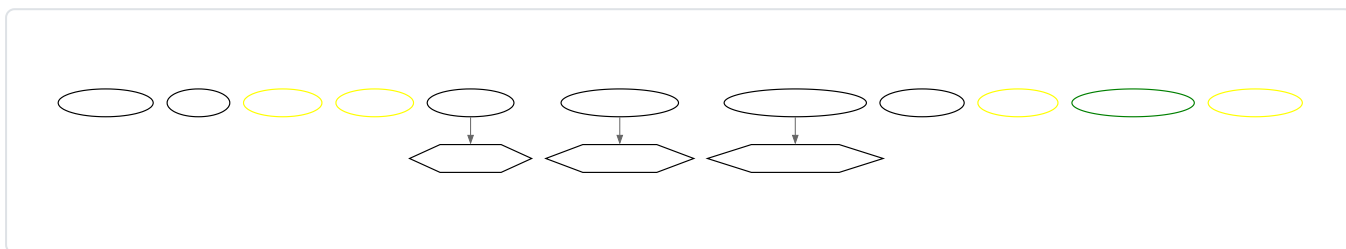
Token Time-Lock Vault

RCPLock is a token locking contract that allows users to lock RCP tokens for a fixed 30-day period. The contract enforces a minimum lock amount of 1000 tokens and provides withdrawal functionality after the lock period expires, with an additional emergency withdrawal option available 90 days after the initial unlock time.

Access Control

none

External Calls



Scan results

lockRCP.sol

Insufficient Balance Validation in emergencyWithdraw Allows DoS

The emergencyWithdraw function does not check the contract's token balance before attempting to transfer tokens to the user. Unlike withdrawTokens which checks the balance and transfers the minimum available, emergencyWithdraw will revert if the

contract has insufficient balance. This creates an asymmetry where normal withdrawals can succeed with partial amounts but emergency withdrawals will fail entirely. An attacker could exploit this by directly transferring tokens out of the contract (if they gain access) or if the contract's balance becomes depleted through other means, effectively preventing users from using emergency withdrawal even after waiting 120 days.

1

Potential Underflow in Fee-on-Transfer Token Handling

In the lockTokens function, the contract calculates the received amount by subtracting balanceBefore from balanceAfter. While Solidity 0.8+ provides automatic overflow protection that would revert on underflow, there's a theoretical edge case where a malicious or buggy token could manipulate balances during the transferFrom call, causing balanceAfter to be less than balanceBefore. This would cause the transaction to revert with an unclear error message, potentially causing DoS for legitimate users trying to lock tokens.

2

lockRCP.sol

Hardcoded Token Address Prevents Contract Reusability

The token address is hardcoded in the constructor as 0x3D81464A248D1DB279E0fF67815c49BDD89Fd20d. This creates several risks: (1) If the token needs to be upgraded or migrated, a new locking contract must be deployed and all users must migrate manually, (2) The contract cannot be reused for other tokens or tested easily on testnets, (3) If the token contract has a critical vulnerability discovered later, there's no way to update the reference. While marked as immutable for gas optimization, this design choice significantly reduces flexibility.

3

lockRCP.sol

Missing Zero Amount Check in withdrawTokens

The withdrawTokens function doesn't explicitly check if the transferAmount is greater than 0 before attempting the transfer. If the contract balance is 0 and a user tries to withdraw, the function will attempt to transfer 0 tokens. While most ERC20 implementations handle zero transfers correctly, some tokens may revert on zero-value transfers, causing unnecessary gas consumption and poor user experience.

Why choose Hashlock?

Trusted by 1,000's of protocols

Auditing some of the world's leading protocols

Comprehensive and formally recognised audit reports

Professional Benefits

Manual code review by security experts

Detailed remediation guidance

Compliance-ready reports & ongoing security partnership

[Get Professional Audit](#)