Liquid Ron
Security Review

JANUARY, 2025

www.chaindefenders.xyz
https://x.com/DefendersAudits

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

Liquid Ron is a Ronin staking protocol that automates user staking actions.

Deposit RON, get liquid RON, a token representing your stake in the validation process of the Ronin Network.

Liquid RON stakes and harvests rewards automatically, auto compounding your rewards and ensuring the best yield possible.

Disclaimer

The Chain Defenders team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Likelihood/Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

Scope

Id	Files in scope
1	ValidatorTracker.sol
2	RonHelper.sol
3	Pausable.sol
4	LiquidRon.sol
5	LiquidProxy.sol
6	Escrow.sol

Roles

Id	Roles
1	Owner
2	User

Executive Summary

Issues found

Severity	Count	Description
High	1	Critical vulnerabilities
Medium	1	Significant risks
Low	1	Minor issues with low impact
Informational	0	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

High

High 01 `operatorFeeAmount` Is Not Deducted From The `totalAssets`

Finding Description and Impact

The `operatorFeeAmount` is not deducted from the `totalAssets` calculation. This omission leads to incorrect computations within the ERC4626 vault. Specifically, the `totalAssets` value includes the operational fees, which results in an overestimation of the vault's assets.

During share withdrawals, this overestimation causes users to receive more assets than they should. Instead of outputting a smaller, correct value, the vault outputs a larger, incorrect value due to the inflated `totalAssets`.

```
function totalAssets() public view override returns (uint256) {
    return super.totalAssets() + getTotalStaked() + getTotalRewards();
}
```

Proof of Concept

Consider the following scenario:

1. Assume the following values:
    - `super.totalAssets()` = 100
    - `getTotalStaked()` = 25
    - `getTotalRewards()` = 5
    - `operatorFeeAmount` = 10
    - `shares` = 10
2. A user attempts to redeem 1 share. The calculation proceeds as follows:
    - `totalAssets` = 100 + 25 + 5 = 130
    - Assets per share = `1 * (130 + 1) / 10` = 13

    However, this calculation is incorrect because the `operatorFeeAmount` is not deducted from `totalAssets`.
3. If the `operatorFeeAmount` were accounted for, the correct calculation would be:
    - `totalAssets` = 100 + 25 + 5 - 10 = 120
    - Assets per share = `1 * (120 + 1) / 10` = 12

    This demonstrates that the current implementation overestimates the value of each share by 1 asset in the current example, based on the totalAssets this can grow exponentially.

Recommended Mitigation Steps

To resolve this issue, deduct the `operatorFeeAmount` from the `totalAssets` calculation. Update the `totalAssets` function as follows:

```
function totalAssets() public view override returns (uint256) {
-     return super.totalAssets() + getTotalStaked() + getTotalRewards()
    ;
+     return super.totalAssets() + getTotalStaked() + getTotalRewards()
     - operatorFeeAmount;
}
```

This change ensures that the `totalAssets` value accurately reflects the vault's assets by excluding the operational fees, thereby preventing the overestimation of share values during withdrawals.

Medium

Mid 01 `onlyOperator` Modifier Does Not Work Correctly

Finding description and impact

The `onlyOperator` modifier in `LiquidRon` is incorrectly implemented. Currently it will not revert only if it's called by an owner who is not set as an operator.

This is due to this check:

```
if (msg.sender ≠ owner() || operator[msg.sender]) revert
    ErrInvalidOperator();
```

Proof of Concept

Let's look at the following cases:

- `msg.sender` is an operator. So their operator mapping is true -> the call reverts.
- `msg.sender` is the owner. Let's say he is not in the mapping of operators -> The call doesn't revert.
- `msg.sender` is the owner who is also set as an operator -> the call reverts.

This means that it will currently allow only the owner who is not set as an operator. By definition it should allow both the owner and the operators.

Recommended Mitigation Steps

Change the given check like this:

```
if (msg.sender ≠ owner() && !operator[msg.sender]) revert
    ErrInvalidOperator();
```

Low

Low 01 Locked Funds

Finding description and impact

In `LiquidRon` both the `deposit` and `receive` functions are missing a call to `_checkUserCanReceiveRon`. This is a problem as it can lead to stuck funds because the given user cannot receive RON back.

```
function deposit() external payable whenNotPaused {
    _depositRONTo(escrow, msg.value);
    Escrow(escrow).deposit(msg.value, msg.sender);
}

receive() external payable {
    if (msg.sender ≠ asset()) {
        _depositRONTo(escrow, msg.value);
        Escrow(escrow).deposit(msg.value, msg.sender);
    }
}
```

Proof of Concept

Let's have the following scenario:

1. User deposits either through the `deposit` or `receive` functions.
2. User cannot receive RON.
3. User tries to withdraw through `requestWithdrawal` but the function reverts leaving his funds stuck.

Recommended Mitigation Steps

Add a call to `_checkUserCanReceiveRon` in the two listed functions.