Protocol Audit Report

Prepared by: Bizarro

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

Bizarro found 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.

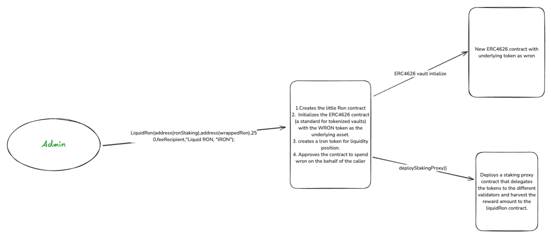
Risk Classification

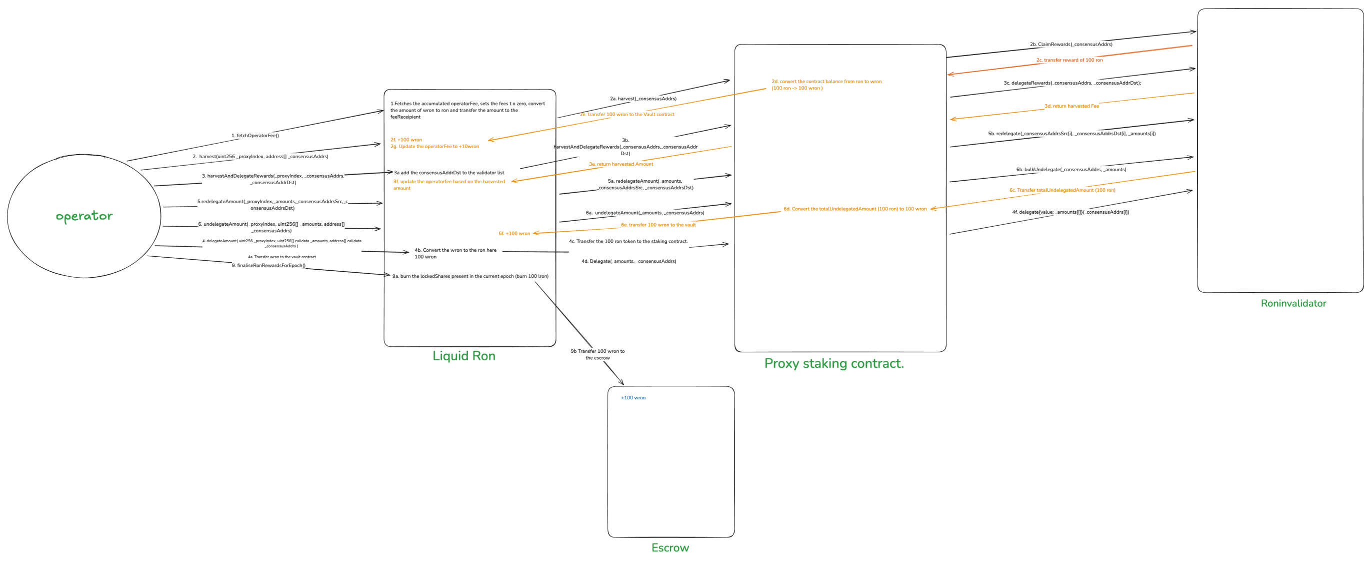
	Impact		
	High	Medium	Low

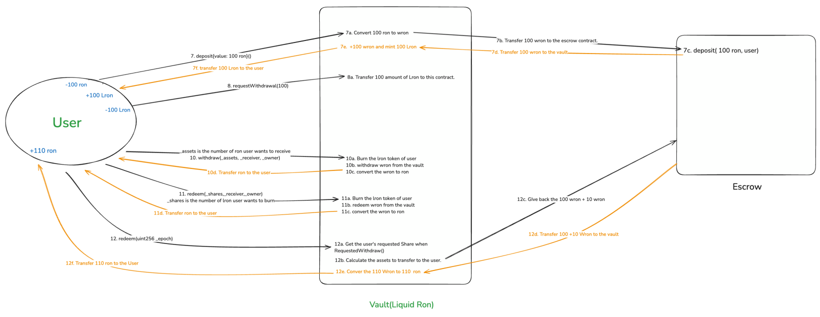
		Impact		
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

WorkFlow By Bizarro


Admin
LiquidRon(address(ronStaking),address(wrappedRon),250,feeRecipient,"Liquid RON, "lRON");
1.Creates the little Ron contract
2. Initializes the ERC4626 contract (a standard for tokenized vaults) with the WRON token as the underlying asset.
3. creates a lron token for liquidity position.
4. Approves the contract to spend wron on the behalf of the caller
ERC4626 vault intialize
New ERC4626 contract with underlying token as wron
deployStakingProxy()
Deploys a staking proxy contract that delegates the tokens to the different validators and harvest the reward amount to the liquidRon contract.


operator
1. fetchOperatorFee()
2. harvest(uint256 _proxyIndex, address[] _consensusAddrs)
3. harvestAndDelegateRewards(_proxyIndex, _consensusAddrs, _consensusAddrDst)
5.redelegateAmount(_proxyIndex,_amounts,_consensusAddrsSrc,_consensusAddrsDst)
6. undelegateAmount(_proxyIndex, uint256[] _amounts, address[] _consensusAddrs)
4. delegateAmount(uint256 _proxyIndex, uint256[] calldata _amounts, address[] calldata _consensusAddrs)
4a. Transfer wron to the vault contract
9. finaliseRonRewardsForEpoch()
1.Fetches the accumulated operatorFee, sets the fees t o zero, convert the amount of wron to ron and transfer the amount to the feeReceipient
2f. +100 wron
2g. Update the operatorFee to +10wron
3a add the consensusAddrDst to the validator list
3f. update the operatorfee based on the harvested amount
6f. +100 wron
4b. Convert the wron to the ron here 100 wron
9a. burn the lockedShares present in the current epoch (burn 100 lron)
Liquid Ron
2a. harvest(_consensusAddrs)
2e. transfer 100 wron to the Vault contract
3b. harvestAndDelegateRewards(_consensusAddrs,_consensusAddrDst)
3e. return harvested Amount
5a. redelegateAmount(_amounts, _consensusAddrsSrc, _consensusAddrsDst)
6a. undelegateAmount(_amounts, _consensusAddrs)
6e. transfer 100 wron to the vault
4c. Transfer the 100 ron token to the staking contract.
4d. Delegate(_amounts, _consensusAddrs)
9b Transfer 100 wron to the escrow
2d. convert the contract balance from ron to wron (100 ron -> 100 wron)
6d. Convert the totalUndelegatedAmount (100 ron) to 100 wron
Proxy staking contract.
2b. ClaimRewards(_consensusAddrs)
2c. transfer reward of 100 ron
3c. delegateRewards(_consensusAddrs, _consensusAddrDst);
3d. return harvested Fee
5b. redelegate(_consensusAddrsSrc[i], _consensusAddrsDst[i], _amounts[i])
6b. bulkUndelegate(_consensusAddrs, _amounts)
6c. Transfer totalUndelegatedAmount (100 ron)
4f. delegate{value: _amounts[i]}(_consensusAddrs[i])
Roninvalidator
+100 wron
Escrow


User
-100 ron
+100 Lron
-100 Lron
+110 ron
7. deposit{value: 100 ron}()
7f. transfer 100 Lron to the user
8. requestWithdrawal(100)
_assets is the number of ron user wants to receive
10. withdraw(_assets, _receiver, _owner)
10d. Transfer ron to the user
11. redeem(_shares,_receiver,_owner)
_shares is the number of lron user wants to burn
11d. Transfer ron to the user
12. redeem(uint256 _epoch)
12f. Transfer 110 ron to the User
7a. Convert 100 ron to wron
7e. +100 wron and mint 100 Lron
8a. Transfer 100 amount of Lron to this contract.
10a. Burn the lron token of user
10b. withdraw wron from the vault
10c. convert the wron to ron
11a. Burn the lron token of user
11b. redeem wron from the vault
11c. convert the wron to ron
12a. Get the user's requested Share when RequestedWithdraw()
12b. Calculate the assets to transfer to the user.
12e. Conver the 110 Wron to 110 ron
Vault(Liquid Ron)
7b. Transfer 100 wron to the escrow contract.
7d. Transfer 100 wron to the vault
7c. deposit(100 ron, user)
12c. GIve back the 100 wron + 10 wron
12d. Transfer 100 +10 Wron to the vault
Escrow

Scope

See scope.txt

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	2
Info	0
Gas	0
Total	3

Findings

Medium

[M-1] Inefficient Loop in `LiquidProxy::harvest` Function Leading to Potential DoS.

Description: When Operator calls the `LiquidRon::harvest` function it calls the `LiquidProxy::harvest` that collects the rewards from the `RoninValidators`. The `LiquidProxy::harvest` function contains an inefficient loop that iterates over an array of consensus addresses (_consensusAddrs). However, the loop is unnecessary because the claimRewards function is called with the entire array in each iteration.

Impact: Potential for Denial of Service (DoS): If the array of consensus addresses is large, the gas cost of the function could exceed the block gas limit, causing the transaction to fail.

Proof of Concept: https://github.com/code-423n4/2025-01-liquid-ron/blob/e4b0b7c256bb2fe73b4a9c945415c3dcc935b61d/src/LiquidProxy.sol#L37

```
    function harvest(address[] calldata _consensusAddrs) external
onlyVault returns (uint256) {
@>        for (uint256 i = 0; i < _consensusAddrs.length; i++) {
            IRoninValidator(roninStaking).claimRewards(_consensusAddrs);
        }
        uint256 claimedAmount = address(this).balance;
        _depositRONTo(vault, claimedAmount);
        return claimedAmount;
    }
```

Recommended Mitigation:

```
    function harvest(address[] calldata _consensusAddrs) external
onlyVault returns (uint256) {
```

```
-         for (uint256 i = 0; i < _consensusAddrs.length; i++) {
              IRoninValidator(roninStaking).claimRewards(_consensusAddrs);
-         }
          uint256 claimedAmount = address(this).balance;
          _depositRONTo(vault, claimedAmount);
          return claimedAmount;
      }
```

Low

[L-1] `LiquidRon::redelegateAmount` does the parameter checks after the implementation, which can cause high gasFee and wrong asset transfers.

Description: The `LiquidRon::redelegateAmount` function calls the `LiquidProxy` contract for redelegating assets to different consensus addresses, the checks for the parameters are introduced after the function implementation.

Proof of Concept:

https://github.com/code-423n4/2025-01-liquid-ron/blob/e4b0b7c256bb2fe73b4a9c945415c3dcc935b61d/src/LiquidRon.sol#L186

Recommended Mitigation:

```
    function redelegateAmount(
        uint256 _proxyIndex,
        uint256[] calldata _amounts,
        address[] calldata _consensusAddrsSrc,
        address[] calldata _consensusAddrsDst
    ) external onlyOperator whenNotPaused {
+         for (uint256 i = 0; i < _consensusAddrsSrc.length; i++) {
+             if (_amounts[i] == 0) revert ErrNotZero();
+             _tryPushValidator(_consensusAddrsDst[i]);
+         }

ILiquidProxy(stakingProxies[_proxyIndex]).redelegateAmount(_amounts,
_consensusAddrsSrc, _consensusAddrsDst);
-         for (uint256 i = 0; i < _consensusAddrsSrc.length; i++) {
-             if (_amounts[i] == 0) revert ErrNotZero();
-             _tryPushValidator(_consensusAddrsDst[i]);
-         }
    }
```

```
        // before                    | 205605           | 205605 | 205605 |
205605 | 1
        // after                     | 28707            | 28707  | 28707  |
28707  | 1
```

[L-2] `LiquidRon::finaliseRonRewardsForEpoch` Transfers ron to the user based on the `lockedSharesPerEpoch[epoch]`, zero amount is not checked before the transfer of the asset which can cause high gas costs.

Proof of Concept: https://github.com/code-423n4/2025-01-liquid-ron/blob/e4b0b7c256bb2fe73b4a9c945415c3dcc935b61d/src/LiquidRon.sol#L245

Recommended Mitigation:

```
 function finaliseRonRewardsForEpoch() external onlyOperator whenNotPaused
{
        uint256 epoch = withdrawalEpoch;
        uint256 lockedShares = lockedSharesPerEpoch[epoch];
        // e before finaliseRonRewardsForEpoch           | 303810
| 303810 | 303810 | 303810 | 1
        // e after  finaliseRonRewardsForEpoch           | 30172
| 30172  | 30172  | 30172  | 1
+       require(lockedShares != 0, "Revert ");
        statusPerEpoch[withdrawalEpoch++] = WithdrawalStatus.FINALISED;
        uint256 assets = previewRedeem(lockedShares);
        _withdraw(address(this), escrow, address(this), assets,
lockedShares);
        lockedPricePerSharePerEpoch[epoch] =
LockedPricePerShare(lockedShares, assets);

        emit WithdrawalProcessFinalised(epoch, lockedShares, assets);
    }
```