Resolv Security Review

Pashov Audit Group

Conducted by: SpicyMeatball, shaflow, aslanbek

May 14th 2025 - May 15th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **resolv-im/resolv-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Resolv Staking

Resolv Staking allows users to stake $RESOLV and receive non-transferable stRESOLV tokens used for governance and time-weighted reward distribution (WAHP). Rewards are calculated forward-only, based on updated stake balances, without retroactive changes. In parallel, the RewardDistributor contract mints and gradually distributes $USR rewards to the stUSR contract using a drip model and rebasing logic, enabling passive, proportionate earning without manual claims.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - fc4b76f5f8739c80a7c3074e6b19bfb090ccf550

fixes review commit hash - 49969c0e55cdf40236c8530d6b4cbbfa4bf780c9

Scope

The following smart contracts were in scope of the audit:

- `ResolvStaking`
- `RewardDistributor`

7. Executive Summary

Over the course of the security review, SpicyMeatball, shaflow, aslanbek engaged with Resolv to review Resolv Staking. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Resolv Staking
Repository	https://github.com/resolv-im/resolv-contracts
Date	May 14th 2025 - May 15th 2025
Protocol Type	Governance token staking

Findings Count

Severity	Amount
Medium	1
Low	2
Total Findings	**3**

Summary of Findings

ID	**Title**	**Severity**	**Status**
[M-01]	withdraw() allows reward claims when claimEnabled is disabled	Medium	Resolved
[L-01]	dripReward can be sandwiched	Low	Acknowledged
[L-02]	Dust _stakingReward affects vesting of large stakingReward	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] `withdraw()` allows reward claims when `claimEnabled` is disabled

Severity

Impact: Medium

Likelihood: Medium

Description

In the `ResolvStaking` contract, the `DEFAULT_ADMIN_ROLE` can set the `claimEnabled` field. When it is set to `false`, the `claim` function for claiming rewards will be disabled.

```
function claim(address _user, address _receiver) external nonReentrant {
        require(claimEnabled, ResolvStakingErrors.ClaimNotEnabled());
        require(msg.sender == _user ||
                msg.sender == usersData[_user].checkpointDelegatee,
            ResolvStakingErrors.InvalidCheckpointCaller());
        checkpoint(_user, true, _receiver, 0);
    }
```

However, in the `withdraw` function, users are allowed to arbitrarily specify whether to claim rewards, and it does **not** check whether `claimEnabled` is disabled.

```
function withdraw(
        bool _claimRewards,
        address _receiver
    ) external nonReentrant {

                 ResolvStakingStructs.PendingWithdrawal storage pendingWithdrawal = us
        uint256 amount = pendingWithdrawal.amount;
        require(amount != 0, ResolvStakingErrors.WithdrawalRequestNotFound());
        require(
          block.timestamp>=pendingWithdrawal.cooldownEnd,
          ResolvStakingErrors.CooldownNotMet
        )

        checkpoint(msg.sender, _claimRewards, _receiver, 0);
        ...
```

This means `claimEnabled` can be bypassed — a user can simply call `initiateWithdrawal` with 1 wei, wait for `cooldownEnd`, and then call `withdraw` with `_claimRewards` set to `true` to successfully claim rewards.

Recommendations

It is recommended that when `claimEnabled` is disabled, `_claimRewards` must be set to `false`.

```
function withdraw(
        bool _claimRewards,
        address _receiver
    ) external nonReentrant {
+       if( _claimRewards && !claimEnabled) revert();
```

8.2. Low Findings

[L-01] `dripReward` can be sandwiched

Every `dripReward` increases stUSR/USR exchange rate, which invites MEV. A bot can:

1. Monitor mempool for dripReward transactions.
2. `deposit` USR into stUSR.
3. After `dripReward` is mined, `withdraw` USR from stUSR, receiving USR rewards without staking.

Recommendations: stUSR#deposit should call `dripReward` at the beginning.

[L-02] Dust `_stakingReward` affects vesting of large `stakingReward`

When `_stakingReward < DRIP_DURATION`, the reward is expected to be released immediately. However, during the `allocateReward` call, any previously unvested tokens will still be released first, and then the new `stakingReward` and `lastCollect` values are set. As a result, the dust reward will only be successfully released on the next `dripReward` call.

```
function dripReward
      (bool _isNewAllocation, uint256 _stakingReward) internal {
        if (_isNewAllocation || _stakingReward < DRIP_DURATION) {
            uint256 balance = IERC20(TOKEN_ADDRESS).balanceOf(address(this));
            if (balance > 0) {
                IERC20(TOKEN_ADDRESS).safeTransfer(ST_USR_ADDRESS, balance);
            }
            drip.stakingReward = _stakingReward;
            drip.lastCollect = block.timestamp;
            emit RewardDripped(balance);
            return;
        }
        ...
```

For example:

1. When there is a large `_stakingReward` in the contract that is still being linearly released.
2. `allocateReward` is called with a dust-sized `_stakingReward`, the previously unreleased large `stakingReward` will be immediately released. 3 While the dust `_stakingReward` will not be released right away — it will only be released after calling the `dripReward` function.

It is recommended that when the input reward amount is a dust value, the protocol should not immediately release the previously unvested `stakingReward`. Instead, it should directly release only the dust amount to avoid disrupting the ongoing linear vesting schedule.