# Defi App Security Review

## Pashov Audit Group

Conducted by: unforgiven, Oblivionis, zark

May 31st 2025 - June 1st 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 **defi-app/defi-app-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

# 4. About Defi App

DefiApp is a platform for managing DeFi assets across chains, offering features like native account abstraction and gasless transactions. The scope was focused on changes is VAirdrop and Airdrop contracts, and RollingAirdrop which allows to airdrop tokens periodically in a "rolling" manner and to set a condition that must be met in order to claim the airdrop.

# 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 hashes:*

- ad0eed6ad1eea90aa34966555f11b01546e294f9
- 4706765931710de87e26d9aa91d45c193aa9c58b

*fixes review commit hash:*

- 27b691c4b55758e394fcc15496d278c3d12e4c60

## Scope

The following smart contracts were in scope of the audit:

- `VAirdrop`
- `Airdrop`
- `MFDDataTypes`
- `MFDLogic`
- `IConditioner`
- `Conditioner`
- `ConditionerS1`
- `RollingAirdrop`

# 7. Executive Summary

Over the course of the security review, unforgiven, Oblivionis, zark engaged with Defi App to review Defi App. In this period of time a total of **4** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Defi App |
| **Repository** | https://github.com/defi-app/defi-app-contracts |
| **Date** | May 31st 2025 - June 1st 2025 |
| **Protocol Type** | Airdrop and Vesting contract |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| Low | 4 |
| **Total Findings** | **4** |

# Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| [L-01] | Function canClaim() doesn't consider contract's balance and pause state | Low | Resolved |
| [L-02] | Airdrop tokens go to msg.sender, not smartAccount recipient | Low | Resolved |
| [L-03] | claimAndStake fails if stakeAmount is under minDLPBalance() | Low | Acknowledged |
| [L-04] | Code should use smartAccount to track claimed amounts | Low | Resolved |

# 8. Findings

## 8.1. Low Findings

### [L-01] Function `canClaim()` doesn't consider contract's balance and pause state

Function `canClaim()` is supposed to return true if user can claim airdrop. It checks multiple things to make sure user can claim airdrop but the issue is that function doesn't check the contract balance. If contract's balance is lower than airdrop amount then user can't claim airdrop tokens. Also It's not possible to claim token when contract is paused and this function doesn't check that if contract is paused or not. It's better to update the `canClaim()` code to check these conditions too or update the comments to clarify this function behavior when balance is low or contract is paused.

### [L-02] Airdrop tokens go to `msg.sender`, not `smartAccount` recipient

The contract includes a `smartAccount` field in the Merkle leaf, which is described as the intended recipient address for the airdrop. However, both the `claim` and `claimAndStake` functions always transfer the airdrop tokens directly to `msg.sender` (the caller), regardless of the `smartAccount` value except when the staking is done.

```
function claim(
      bytesmemoryleafElements,
      bytes32[]calldatamerkleProof
    ) external whenNotPaused nonReentrant {
        // ...

        // Transfer the tokens
@>        distributionToken.safeTransfer(msg.sender, claimableAmount);
        emit AirdropRewardsClaim(msg.sender, claimableAmount, globalAmount);
    }
```

# [L-03] `claimAndStake` fails if `stakeAmount` is under `minDLPBalance()`

In the staking flow, the `RollingAirdrop` contract may require users to use `claimAndStake` (when `condition` is true). However, if the staked amount is less than `IBountyManager($.bountyManager).minDLPBalance()`, the staking operation in `MFDLogic.stakeLogic` will revert indefinitely due to this line:

```
function stakeLogic(
        MultiFeeDistributionStorage storage $,
        uint256 _amount,
        address _onBehalf,
        uint256 _typeIndex,
        bool _isRelock
    ) public {
        if (_amount == 0) return;
        if ($.bountyManager != address(0)) {
@>            if (_amount < IBountyManager($.bountyManager).minDLPBalance
  ()) revert MFDLogic_invalidAmount();
        }
```

This means a user who is only allowed to use `claimAndStake` for their claim may be unable to receive their airdrop if the amount is too low to satisfy the staking contract's minimum requirement, and their entire claim will revert.

Consider adding logic in the `claimAndStake` function of the airdrop contract to check, before attempting to stake, whether the stake amount is at least the minimum required by the staking contract.

# [L-04] Code should use `smartAccount` to track claimed amounts

Code keeps track of the users claimed airdrop amounts in variable `claimedAmounts[]`. The issue is that function `claim()` and `claimAndStake()` use `claimedAmounts[msg.sender]` to check for user claimed amounts instead of using `claimedAmounts[smartAccount]`. Users account in the system is `smartAccount` and users can register multiple address for their smart account. As airdrop belongs to the `smartAccount` code should keep track of user's claimed amounts by using `smartAccount` address. In current implementation, if user receive two airdrop based on the `smartAccount` and different wallet address then claimable calculation would be wrong and user would receives up

to twice amount of airdrop as `claimedAmounts[msg.sender]` won't show claimed amount for the second wallet address.

Recommendations: Use `smartAccount` to keep track of user claimed amounts in all places.