RENASCENCE

# Arcadia Finance Audit Report

Version 1.0


Audited by:

**zzykxx**

**0xadrii**

**bytes032**


June 27, 2024

# Contents

# 1 Introduction

## 1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

## 1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

## 1.3 Risk Classification

| | **Impact: High** | **Impact: Medium** | **Impact: Low** |
|---|---|---|---|
| **Likelihood: High** | High | High | Medium |
| **Likelihood: Medium** | High | Medium | Low |
| **Likelihood: Low** | Medium | Low | Low |

### 1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

### 1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

# 2 Executive Summary

## 2.1 About Arcadia Finance

Arcadia is a non-custodial platform for asset management focused on unlocking the full potential of DeFi.

The platform gives users a single on-chain account to access margin, compose a portfolio of multiple yield sources, and improve capital efficiency across underlying DeFi protocols.

## 2.2 Overview

| | |
|---|---|
| Project | Arcadia Finance |
| Repository | accounts-v2, asset-manager, lending-v2 |
| Commit Hash | 068f297cbbf1..., 08725871e68a..., a83dd0b94422... |
| Mit Hash | c1200c256ac5..., f34ae7048405..., 88c1b64f9059... |
| Date | 24 June 2024 - 26 June 2024 |

## 2.3 Issues Found

| **Severity** | **Count** |
|---|---|
| High Risk | 0 |
| Medium Risk | 0 |
| Low Risk | 3 |
| Informational | 0 |
| **Total Issues** | **3** |

# 3 Findings Summary

| ID | Description | Status |
|---|---|---|
| L-1 | `executeAction()` could revert on zero amount transfers | Resolved |
| L-2 | Depositing NFT liquidity positions that already collected fees leads to tokens being stuck | Resolved |
| L-3 | `TrancheWrapper` will be undeployable for some ERC20 tokens | Resolved |

# 4 Findings

## Low Risk

### [L-1] `executeAction()` could revert on zero amount transfers

**Context:**

- UniswapV3Compounder#L219
- UniswapV3Compounder#L220

**Description:** The function `UniswapV3Compounder::executeAction()` is meant to transfer any tokens left in the contract to the `initiator` as a form of compensation for calling `UniswapV3Compounder::compoundFees()`:

```
function executeAction(bytes calldata compoundData) external override returns
(ActionData memory assetData) {
  ...
  // Initiator rewards are transferred to the initiator.
  ERC20(position.token0).safeTransfer(initiator,
  ERC20(position.token0).balanceOf(address(this)));
  ERC20(position.token1).safeTransfer(initiator,
  ERC20(position.token1).balanceOf(address(this)));
  ...
}
```

It's possible that the amount of `token0` and/or `token1` left in the contract is `0`, which would make the whole call revert in case `token0` and/or `token1` is a token that reverts on `0` transfers.

**Recommendation:** Execute the transfer only if the balance in the contract is greater than `0`.

### [L-2] Depositing NFT liquidity positions that already collected fees leads to tokens being stuck

**Context:**

- StakedSlipstream.sol#L412

**Description:** The function `StakedSlipstream::mint()` calls `CLGauge::deposit()` in order to deposit an NFT liquidity position in the relative gauge. If the NFT liquidity position `tokensOwed0`/`tokensOwed1` state variables are not `0`, meaning the position already accumulated fees and/or liquidity was decreased, `CLGauge::deposit()` will trigger a downstream call to `CLPool::collect()` which will transfer `token0`/`token1` to the caller, in this case the `StakedSlipstream` contract.

The `StakedSlipstream::mint()` function doesn't take this possibility in consideration resulting in any tokens sent to it to be locked in the `StakedSlipstream` contract.

**Recommendation:** The intended usage is for users to mint an NFT position and then deposit it immediately, for this reason a good solution is to prevent NFT positions whose `tokensOwed0`/`tokensOwed1` are not `0` from being deposited.

### [L-3] `TrancheWrapper` will be undeployable for some ERC20 tokens

**Context:**

- TrancheWrapper.sol#L44-L45

**Description:** Currently, when deploying a `TrancheWrapper` a call to the tranche's `asset` `name` and `symbol` methods will be performed so that the correct name and symbol can be set to the wrapper. Although `name` and `symbol` expect a `string` to be returned, some weird ERC20 tokens such as MKR return a `bytes32` value for such data. This will prevent tranche wrappers from being deployed for this subset of ERC20 tokens.

**Recommendation:** Pass the desired `name`/`symbol` in the `TrancheWrapper`'s constructor so that these type of tokens can be directly supported:

```
-constructor(address tranche)
+constructor(address tranche, string memory name, string memory symbol)
        ERC4626(
            ERC4626(tranche).asset(),
-            string(abi.encodePacked("Wrapped ", ERC4626(tranche).asset().name())),
-            string(abi.encodePacked("w", ERC4626(tranche).asset().symbol()))
+            string(abi.encodePacked("Wrapped ", name)),
+            string(abi.encodePacked("w", symbol))
        )
    {
        TRANCHE = tranche;
        LENDING_POOL = ITranche(tranche).LENDING_POOL();
    }
```

**Arcadia:** We should wrap the underlying Trache, not the underlying asset. So the `.asset` should not have been called in the first place. Fixed in PR-165.

**Renascence:** The issue has been fixed by fetching the `name` and `symbol` for `TrancheWrapper` directly from the tranche, instead of the tranche's asset.