Ambit Finance Audit Report

Version 1.1

Conducted by: Kiki

April 22, 2024

Table of Contents

1 Introduction

1.1 About Kiki

I'm Kiki, currently operating as an Security Researcher. My work includes performing security reviews with one of the best firms, Guardian Audits, finding bugs in live contracts through bug bounties, or working directly with protocols via private audits. For inquiries, you can reach me through Twitter or Telegram.

1.2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

1.3 Risk classification

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

1.3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

1.3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

1.3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

1.3.4 Informational findings

Informational findings will also be included. These encompass recommendations to enhance code style, operations alignment with industry best practices, gas optimizations, adherence to documentation and EIP standards, and overall contract design.

Informational findings typically have minimal impact on code functionality or security risk.

Evaluating all vulnerability types, including informational ones, is crucial for a comprehensive security audit to ensure robustness and reliability.

2 Executive Summary

2.1 Ambit Finance

Ambit is cutting-edge, cross-chain DeFi protocol offering users simple yields on stablecoin deposits, sustainable money market lending, and risk-defined portfolio investment strategies - all within a user-friendly environment.

More on the protocol can be found on the official documentation page.

2.2 Overview

Project Name	Ambit Finance
Codebase	ambitfi-contracts
Operating platform	Arbitrum One, BNB Chain
Language	Solidity
Audited commits	[1] 2116e63366c794ef6bf5e0bbdb6e20b2b6a5b2c0 [2] 916613307d3b1739f01b4af3aa5fb90b3ce6cf06
Remediation commit	Pending
Audit methodology	Manual Review

2.3 Scope

Files and folders in scope

- contracts/protocol/market/MarketLiquidation.sol
- contracts/protocol/market/Market.sol
- contracts/protocol/market/Liquidator.sol
- contracts/protocol/lens/AccountLens.sol
- contracts/protocol/portfolio/YieldBearingCustodian.sol
- contracts/protocol/vault/DepositorVault.sol
- contracts/protocol/vault/YieldVault.sol

2.4 Issues Found

Severity	Total Found	Resolved	Partially Resolved	Acknowledged
Critical risk	0	0	0	0
High risk	0	0	0	0
Medium risk	0	0	0	0
Low risk	2	1	0	1

2.5 Findings & Resolutions

ID	Title	Severity	Status
L-01	liquidateTotalSupply can lead to excessive liquidations	Low	Acknowledged
L-02	Missing explicit zero share check in onwithdrawInternal	Low	Resolved

3 Findings

3.1 Low Risk

3.1.1 liquidateTotalSupply can lead to excessive liquidations

Severity: *Low risk (Acknowledged)*

Context: [⊶1] : ***MarketLiquidation.sol***:*189*

Description

When the `liquidateTotalSupply` function is called the caller can specify an amount they would like to liquidate. If the account is above the `_smallAccountThreshold` the most the caller can liquidate is 50% of the liabilities. This provides some assurance for users of the protocol that if they have a balance above the `_smallAccountThreshold` they will be able to keep a portion of their collateral. This feature is not allotted to accounts who are below the `_smallAccountThreshold`, for these users there is no cap to the amount that can be liquidated.

The potential issue here is that if a user is slightly above the `_smallAccountThreshold` they can end up having greater than 50% of their funds liquidated. This can happen when the amount specified through the `liquidateTotalSupply` function is large enough to move the account from above the `_smallAccountThreshold` to below, as well as the specified amount being small enough so that the account is still unhealthy and eligible for another liquidation.

When this happens the user can be liquidated again by anyone through the `liquidator` contract for the full amount of remaining liabilities.

Recommendation

Consider checking that the function is indeed healthy after a being liquidated through the `liquidateTotalSupply` function.

It is also worth noting that because liquidating through the `liquidateTotalSupply` function is a permissioned feature and cannot be abused by arbitrary users it may only be necessary for the protocol to monitor this and ensure they are passing in an adequate amount for `context.params.totalSupply`.

Resolution: Acknowledged by the team.

Because liquidateTotalSupply is permissioned, the protocol will monitor this edge case.

3.1.2 Missing explicit zero share check in onwithdrawInternal

Severity: *Low risk (Resolved)*

Context: [⊶2] : ***DepositorVault.sol***:*466*

Description

When a user goes to withdraw assets from the `DepositorVault` contract they have two options to do so. They can call the `redeem` function or the `withdraw` function the difference between the two is

primarily that `redeem` function takes a `shares` amount from the user, while `withdraw` takes an asset `amount` from the user.

When calling the `redeem` function there is an explicit check that the inputted `shares` is non-zero. Then when the `withdrawInternal` function is called there is another explicit check that the asset `amount` is non-zero. By doing this there is a guarantee that the user cannot withdraw a non-zero amount for zero shares.

This explicit check pattern however is not present when withdrawing through the `withdraw` function. When this function is called there is an explicit check that the asset `amount` is non-zero, and then when the `withdrawInternal` function is called, `amount` is again checked that it is non-zero. Throughout this there is no explicit check that `shares` is non-zero.

With everything working as intended there is no feasible path to withdraw a non-zero amount of assets while burning zero `shares`, however there is also no explicit check preventing this incase the protocol has unexpected behavior in the future.

Recommendation

Consider explicitly checking that `shares` is also non zero in the `withdrawInternal` function.

Resolution: Resolved. Fix was implemented in .

Check for 0 shares was added.