Lista DAO: Moolah

Security Review

Cantina Managed review by:

Sujith Somraaj, Security Researcher
RustyRabbit, Security Researcher

November 12, 2025

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**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Fixed position over-repayment causes cross-position accounting desynchronization . . . 4
  - 3.1.2 Liquidation resets user interest regardless of the amount repaid and collateral seized 5
  - 3.1.3 Normailzation/denormalization rounding errors can lead to state inconsistencies causing reverts . . . 8
  - 3.1.4 Converting dynamic to fixed position allows users to avoid paying interest on the dynamic position . . . 9
- 3.2 Medium Risk . . . 10
  - 3.2.1 Missing `pause` and `unpause` functions in `LendingBroker.sol` . . . 10
- 3.3 Low Risk . . . 11
  - 3.3.1 Moolah should check the broker's market Id when setting a new broker . . . 11
  - 3.3.2 Amount to shares rounding can lead to small overpayments . . . 11
  - 3.3.3 Complete protocol DoS when rate accrual calculation overflows . . . 11
  - 3.3.4 Centralization risk in `refinanceMaturedFixedPositions()` . . . 14
  - 3.3.5 Missing pause functionality . . . 14
- 3.4 Informational . . . 15
  - 3.4.1 Remove unused file imports . . . 15
  - 3.4.2 Redundant state variable initialization in `LendingBroker` initializer . . . 15
  - 3.4.3 `DOMAIN_SEPARATOR` immutable variable uses wrong address . . . 15
  - 3.4.4 Loan token allowance from broker to Moolah is not reset when not used . . . 16
  - 3.4.5 `repayPrincipalAmt` variable shadowing in `_deductFixedPositionsDebt()` . . . 16
  - 3.4.6 User repaying a fixed position early is paying a penalty on the penalty itself . . . 16
  - 3.4.7 Incompatible use of immutable variables in UUPS upgradeable contract . . . 17
  - 3.4.8 Increase test coverage . . . 17
  - 3.4.9 Use consistent naming convention . . . 17
  - 3.4.10 Replace magic numbers with constants . . . 18
  - 3.4.11 Redundant return in `setRatePerSecond()` function . . . 18
  - 3.4.12 Declare all `BrokerMath` and `PriceLib` library functions internal . . . 19
  - 3.4.13 Fix typos . . . 19

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 Sep 29th to Oct 5th the Cantina team conducted a review of moolah on commit hash 51e1adcf. The team identified a total of **23** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	4	4	0
Medium Risk	1	1	0
Low Risk	5	3	2
Gas Optimizations	0	0	0
Informational	13	7	6
Total	**23**	**15**	**8**

3 Findings

3.1 High Risk

3.1.1 Fixed position over-repayment causes cross-position accounting desynchronization

Severity: High Risk

Context: LendingBroker.sol#L282

Description: The `LendingBroker.repay()` function for fixed positions has an accounting flaw where any excess repayment is applied to the user's total Moolah debt but does not update the balances of the user's other positions, including dynamic and other fixed ones.

When a user repays more than the remaining principal of a fixed position, the extra amount is used to pay off debt at the Moolah level, which consolidates all positions. However, the broker does not reduce the balances of these affected positions, resulting in a continued mismatch between the broker's records and Moolah's actual debt balances.

Proof of Concept: The following test demonstrates how overpaying fixed position, paid off dynamic position debt in the Moolah contract but messes up the LendingBroker's internal account:

```
function test_audit() external {
 _supplyCollateralToMoolah();

 vm.startPrank(BORROWER);
 broker.borrow(80e18);
 broker.borrow(20e18, 1);
 moolah.position(id, BORROWER);

 broker.userFixedPositions(BORROWER);
 loanToken.setBalance(BORROWER, 200e18);
 loanToken.approve(address(broker), 200e18);

 vm.warp(block.timestamp + 100 seconds);

 broker.repay(80e18, 1, BORROWER);
 broker.userFixedPositions(BORROWER);
 broker.userDynamicPosition(BORROWER);
}

 /// test helpers
 function _supplyCollateralToMoolah() internal {
   collateralToken.setBalance(BORROWER, 100e18);

   vm.startPrank(BORROWER);
   collateralToken.approve(address(moolah), 100e18);
   moolah.supplyCollateral(marketParams, 100e18, BORROWER, bytes(""));
   vm.stopPrank();
 }
}
```

Likelihood explanation: The vulnerability requires no special privileges or unusual market conditions; it only requires a repayment amount larger than the specific position being repaid. Given that users commonly have multiple positions and often prefer to over-repay to avoid dust amounts, this scenario has a moderate-to-high likelihood of occurring naturally.

Impact explanation: Since the issue causes broken accounting, the impact is considered HIGH.

Recommendation: Consider capping the principal amount at the fixed position value to resolve the issue.

```
  uint256 repayInterestAmt = amount < accruedInterest ? amount : accruedInterest;
- uint256 repayPrincipalAmt = amount - repayInterestAmt;
+ uint256 amountLeft = amount - repayInterestAmt;
+ uint256 repayPrincipalAmt = amountLeft > position.principal ? position.principal : amountLeft;
```

Note: the converse issue is also true, where the dynamic `repay()` function could be used to pay for a fixed loan, affecting the entire internal accounting of the `LendingBroker.sol` contract.

Lista DAO: Fixed in commit e44f7dff.

Cantina Managed: Fix verified.

3.1.2 Liquidation resets user interest regardless of the amount repaid and collateral seized

Severity: High Risk

Context: LendingBroker.sol#L434-L458

Summary: When partially liquidating a position the interest is not fully collected and accounted for incorrectly. This will result in desynched tracking of principal between broker and Moolah and can lead to loss of interest for the protocol or excessive seizing of collateral for the user.

Finding Description: When liquidating with an amount collateral seized less than the full amount of collateral only part of the interest due at the broker is reclaimed. This is because the the interest due is averaged out over the complete collateral. So seizing the collateral partially (and repaying the discounted price to Moolah) only seizes the interest partially.

However at the broker the difference between the debt at the broker and the debt at Moolah is regarded as repayed. This is not the case as as this difference includes the full amount of the interest while only part of it was repayed.

Furthermore the reduction of the share price at Moolah is averaged over the interest of the dynamic and fixed positions. At the broker however this averaged amount is first used to account for both the interest and Principal of the dynamic position and only then for the interest and principal of the fixed positions (each individually). If Interest/principal ratios of different positions vary, which they undoubtedly will do, the principal accounting can become desynched between broker and Moolah.

Impact Explanation: Desynchronisation of Principal tracking combined with incorrectly distributing funds is regarded as high.

Likelihood Explanation: Liquidation is limited to the protocol controlled liquidator but even if the correct amount of collateral to be seized can be determined the accounting differences will occur. As such the likelihood is considered high.

Proof of Concept: Following is a test that shows the effect of partial liquidations.

Note: this test frequently runs in the rounding issue described in finding "Normailzation/denormalization rounding errors can lead to state inconsistencies causing reverts". In order to run it modify the repay() to use `zeroFloorSub()` to avoid the issue.

```
function test_liquidation_before_repay() public {
    address liquidator2 = address(0xABC);
    oracle.setPrice(address(BTCB), 100_000e8);  //set price to 100k for easier reading

    vm.prank(BOT);
    rateCalc.setRatePerSecond(address(broker), RATE_SCALE +  30300 * 10**14 );       //should be about 10% APR

    vm.startPrank(borrower);
    moolah.withdrawCollateral(marketParams, COLLATERAL, borrower, ADMIN);            //withdraw everything for
    ↪  easier reading
    BTCB.setBalance(borrower, 1 ether);
    _logBalances("      BEFORE SUPPLY");
    moolah.supplyCollateral(marketParams, 1 ether, borrower, "");
    _logBalances("AFTER SUPPLY / BEFORE BORROW");
    uint256 borrowAmt = 80_000 ether;                                                //lltv is 80% so borrow the
    ↪  limit
    broker.borrow(borrowAmt);
    vm.stopPrank();

    skip(365 days);
    uint256 rate = IRateCalculator(address(rateCalc)).accrueRate(address(broker));    //update the rate after
    ↪  each time warp

    (uint256 principalBefore, uint256 normalizedBefore) = broker.dynamicLoanPositions(borrower);
    uint256 actualDebt = BrokerMath.denormalizeBorrowAmount(normalizedBefore, rate);
    uint256 interestOutstanding = actualDebt > principalBefore ? actualDebt - principalBefore : 0;
    assertGt(interestOutstanding, 0, "no accrued interest");


    //uint256 baseAmount = (actualDebt - principalBefore) / 10 ;  //only liquidate based on partial interest
    ↪  shows liquidator only gets part of the interest due and borrower loses less
    uint256 baseAmount = principalBefore /3 ;            //partial repayment of the principal during liquidation
```

```
    //uint256 baseAmount = principalBefore ;                //full repayment of principal, borrower does not need
    ↪  to repay and can claim the remaining collateral. This effectively transfers the value of the interest to
    ↪  the liquidator, repays the principal at moolah and the borrowers pays the cost including the liquidation
    ↪  incentive.
    //uint256 baseAmount = actualDebt;                      //trying to liquidate for the full actual debt fails
    ↪  because the amount of shares calculated based on the actual debt and the borrow assets and shares at
    ↪  Moolah leads an overestimtation of the shares leading to a revert in the claim() due to underflow (as
    ↪  documented in the function natspec)).
    Market memory marketBefore = moolah.market(id);
    uint256 repaidShares = baseAmount.toSharesDown(
      marketBefore.totalBorrowAssets,
      marketBefore.totalBorrowShares
    );

    Position memory userPos = moolah.position(id, borrower);
    console.log("User position at Moolah borrowShares: %s, repaidShares based on user posisition in broker:
    ↪  %s", userPos.borrowShares, repaidShares);

    uint256 startingBalance = repaidShares.toAssetsUp(marketBefore.totalBorrowAssets,
    ↪  marketBefore.totalBorrowShares);
    LISUSD.setBalance(liquidator2, startingBalance);

    /// LIQUIDATION
    _logBalances("      BEFORE LIQUIDATION");
    vm.startPrank(liquidator2);
    IERC20(address(LISUSD)).approve(address(moolah), type(uint256).max);
    moolah.liquidate(marketParams, borrower, 0, repaidShares , bytes(""));
    vm.stopPrank();

    Position memory remainingPosition = moolah.position(id, borrower);
    //console.log("borrowShares %s",remainingPosition.borrowShares);
    //console.log("collateral %s",remainingPosition.collateral);
    //console.log("supplySharess  %s",remainingPosition.supplyShares);

    _logBalances("        AFTER LIQUIDATION");

    //BORROWER REPAYS THE REST
    vm.startPrank(borrower);
    (uint256 principalToRepay, uint256 normalizedToRepay) = broker.dynamicLoanPositions(borrower);
    if (principalToRepay > 0 ) {
      uint256 repayAmt = BrokerMath.denormalizeBorrowAmount(normalizedToRepay, rate);
      uint256 actualRepayAmt = repayAmt > principalToRepay ? repayAmt : principalToRepay;             //
      ↪  denormalized amount of positionshould always be higher but sometimes it isn't du to rounding so we take
      ↪  the complete principal in this case to get everything out.
      console.log("denormalized actual debt: %s, principal: %s. Choosing to repay %s", repayAmt/10**18,
      ↪  principalToRepay/10**18, actualRepayAmt/10**18);
      broker.repay(actualRepayAmt, borrower);     //this revertsdu to underflow
    } else {
      console.log("liquidation paid for the principal. No need to repay, just withdraw");
    }

    _logBalances("       AFTER REPAYMENT");

    vm.startPrank(borrower);
    remainingPosition = moolah.position(id, borrower);
    //console.log("borrowShares %s",remainingPosition.borrowShares);
    //console.log("collateral %s",remainingPosition.collateral);
    //console.log("supplySharess  %s",remainingPosition.supplyShares);
    moolah.withdrawCollateral(marketParams, remainingPosition.collateral, borrower, borrower);
    vm.stopPrank();

    _logBalances("       AFTER WITHDRAWAL");
    //console.log("Actual debt for borrower before liquidation %s",actualDebt/10**18 );
    console.log("Interest due by borrower before liquidation: %s",(actualDebt - principalBefore)/10**18);
    console.log("cost of liquidation to liquidator: %s",startingBalance/10**18);

  }
```

For reference this is the logs for a liquidation 1/3rd of the principal.

```
 ---------------------------------
|          BEFORE SUPPLY          |
|---------------------------------|
|   borrower |  Moolah  | liquid. |
|----------- |--------- |-------- |
|        0   | 1000000  |    0    |
```

```
|---------------------------------|
|         COLLATERAL VALUE        |
|----------- |--------- |-------- |
|   100000   |    0     |    0    |
|---------------------------------|
|           TOTAL VALUE           |
|----------- |--------- |-------- |
|   100000   | 1000000  |    0    |
|_________________________________|

 ---------------------------------
|   AFTER SUPPLY / BEFORE BORROW  |
|---------------------------------|
|   borrower |  Moolah  | liquid. |
|----------- |--------- |-------- |
|      0     | 1000000  |    0    |
|---------------------------------|
|         COLLATERAL VALUE        |
|----------- |--------- |-------- |
|     0      | 100000   |    0    |
|---------------------------------|
|            TOTAL VALUE          |
|----------- |--------- |-------- |
|     0      | 1100000  |    0    |
|_________________________________|

User position at Moolah borrowShares: 80000000000000000000000000000, repaidShares based on user posisition in
↪  broker: 26666666666666666666666000000

 ---------------------------------
|        BEFORE LIQUIDATION       |
|---------------------------------|
|   borrower |  Moolah  | liquid. |
|----------- |--------- |-------- |
|   80000    |  920000  |  26666  |
|---------------------------------|
|         COLLATERAL VALUE        |
|----------- |--------- |-------- |
|     0      | 100000   |    0    |
|---------------------------------|
|       TOTAL VALUE               |
|----------- |--------- |-------- |
|   80000    | 1020000  |    0    |
|_________________________________|

 ---------------------------------
|         AFTER LIQUIDATION       |
|---------------------------------|
|   borrower |  Moolah  | liquid. |
|----------- |--------- |-------- |
|   80000    |  946666  |    0    |
|---------------------------------|
|          COLLATERAL VALUE       |
|----------- |--------- |-------- |
|     0      |  69157   |  30842  |
|---------------------------------|
|            TOTAL VALUE          |
|----------- |--------- |-------- |
|   80000    | 1015823  |  30843  |
|_________________________________|

denormalized actual debt: 53333, principal: 53333. Choosing to repay 53333

 ---------------------------------
|         AFTER REPAYMENT         |
|---------------------------------|
|   borrower |  Moolah  | liquid. |
|----------- |--------- |-------- |
|   26666    | 1000000  |    0    |
|---------------------------------|
|          COLLATERAL VALUE       |
|----------- |--------- |-------- |
|     0      |  69157   |  30842  |
|---------------------------------|
|             TOTAL VALUE         |
|----------- |--------- |-------- |
```

```
|   26666    | 1069157  |  30843  |
|_____________________________________|

 _____________________________________
|         AFTER WITHDRAWAL            |
|_____________________________________|
|   borrower |  Moolah   | liquid. |
|----------- |---------- |-------- |
|   26666    | 1000000   |    0    |
|_____________________________________|
|           COLLATERAL VALUE          |
|----------- |---------- |-------- |
|   69157    |    0      |  30842  |
|_____________________________________|
|             TOTAL VALUE             |
|----------- |---------- |-------- |
|   95823    | 1000000   |  30843  |
|_____________________________________|

Interest due by borrower before liquidation: 8021
cost of liquidation to liquidator: 26666
```

Recommendation: The amount of principal and interest should be indicated separately when Moolah calls `liquidate()` on the broker instead of deducing the total amount repayed from the accounting differences between the broker and Moolah. This way the correct amounts can be subtracted from the interest and principal at the broker.

Additionally consider a method of liquidating where seizing all of the user's collateral isn't needed. This could be done for example by averaging the reduction of collateral price over the amount seized rather than the full amount of collateral.

Lista DAO: Fixed in commit 247299b7.

Cantina Managed: Fixed.

3.1.3 Normailzation/denormalization rounding errors can lead to state inconsistencies causing reverts

Severity: High Risk

Context: LendingBroker.sol#L236, LendingBroker.sol#L250

Summary: Normailzation and denormalization cause rounding errors to occur. When these values are used in subtractions (directly or indirectly) they can lead to underflows thereby reverting on critical functions.

Finding Description: In most places where normalized and denormalized values are used `FixPointMathLib`'s `ZeroFloorSub()` is used to avoid underflows.

However in some places plain subtraction is still used in combination with rounded values either directly computed or from previous calcucations that caused state changes. One of those places is in repay() where the `accruedInterest` is calculated by subtracting the previously stored principal from the position's denormalized `normalizedDebt`.

Impact Explanation: The impacted areas include critical functions like repay and liquidate. As such the impact is deemed high.

Likelihood Explanation: Although the issue needs specific values to occur they can occur naturally and is purely a matter of chance. AS such the likelihood is regarded as high.

Proof of Concept: Following test although not targeted specifically to this issue shows that it can occur rather easily.

```
function test_repay_rounding_reverts() public {
    oracle.setPrice(address(BTCB), 100_000e8);  //set price to 100k for easier reading

    vm.prank(BOT);
    rateCalc.setRatePerSecond(address(broker), RATE_SCALE +  30300 * 10**14 );       //should be about 10% APR

    vm.startPrank(borrower);
    moolah.withdrawCollateral(marketParams, COLLATERAL - 1 ether, borrower, ADMIN);  //withdraw everything
    ↪  except 1 BTCB and send it to admin (so borrower's balance is clear)
```

```
    uint256 borrowAmt = 80_000 ether;                                    //lltv is 80% so borrow the
    ↪  limit
    broker.borrow(borrowAmt);
    vm.stopPrank();

    skip(365 days);
    uint256 rate = IRateCalculator(address(rateCalc)).accrueRate(address(broker));

    (uint256 principalBefore, uint256 normalizedBefore) = broker.dynamicLoanPositions(borrower);
    uint256 actualDebt = BrokerMath.denormalizeBorrowAmount(normalizedBefore, rate);
    uint256 interestOutstanding = actualDebt > principalBefore ? actualDebt - principalBefore : 0;
    assertGt(interestOutstanding, 0, "no accrued interest");

    Market memory marketBefore = moolah.market(id);
    uint256 repaidShares = principalBefore.toSharesDown(
      marketBefore.totalBorrowAssets,
      marketBefore.totalBorrowShares
    );
    repaidShares = repaidShares *20/100; // Partially liquidate

    address liquidator2 = address(0xABC);
    uint256 startingBalance = repaidShares.toAssetsUp(marketBefore.totalBorrowAssets,
    ↪  marketBefore.totalBorrowShares);
    LISUSD.setBalance(liquidator2, startingBalance);

    /// LIQUIDATION
    vm.startPrank(liquidator2);
    IERC20(address(LISUSD)).approve(address(moolah), type(uint256).max);
    moolah.liquidate(marketParams, borrower, 0, repaidShares , bytes(""));
    vm.stopPrank();

    //Borrower repays the rest
    vm.startPrank(borrower);
    (uint256 principalToRepay, uint256 normalizedToRepay) = broker.dynamicLoanPositions(borrower);
    uint256 repayAmt = BrokerMath.denormalizeBorrowAmount(normalizedToRepay, rate);
    broker.repay(repayAmt, borrower);     //this reverts due to underflow
    vm.stopPrank();
}
```

Recommendation (optional): Ensure all places where rounded values are used subtractions use `ZeroFloorSub()`. Additionally add sanity checks that certain state changes cannot occur, for instance a position having a Principal that is larger than the denormalized `normalizedDebt` as in the provided test case.

Lista DAO: Fixed, all values derived by rounding, using `zeroFloorSub` when doing subtractions since commit aac416d1.

Cantina Managed: Fixed.

3.1.4 Converting dynamic to fixed position allows users to avoid paying interest on the dynamic position

Severity: High Risk

Context: LendingBroker.sol#L383-L410, LendingBroker.sol#L822-L836

Summary: When converting a dynamic position to a fixed position accounting discrepencies between broker and Moolah leads to Moolah refunding the dynamic interest when repaying the fixed position.

Finding Description: A dynamic position tracks the principal and interest at the broker while also tracking (the same) principal at Moolah.

When converting a dynamic position to a fixed one the interest at the broker is converted into principal at the broker without adding principal to the position at Moolah. This creates a discrepancy in the principal accounting between broker and Moolah. When repaying the principal at Moolah only the initial principal is taken and the remaining part (the interest from the dynamic position is refunded).

Impact Explanation: The impact is considered high as users can apply this to any dynamic position and thus avoid paying the interest.

Likelihood Explanation: As no special situation or access control is required the likelihood is high.

Proof of Concept: Following test shows that after converting a dynamic position to a fixed one, the borrower can withdraw their collateral after repaying only the original principal.

```
function test_convertDynamicToFixed_noInterest() public {
    vm.prank(MANAGER);
    broker.setFixedTermAndRate(52, 30 days, RATE_SCALE + 1);

    uint256 borrowAmt = 750 ether;
    vm.prank(borrower);
    broker.borrow(borrowAmt);

    vm.prank(MANAGER);
    rateCalc.setMaxRatePerSecond(address(broker), RATE_SCALE + 6);
    vm.prank(BOT);
    rateCalc.setRatePerSecond(address(broker), RATE_SCALE + 4);
    skip(3 days);

    (uint256 principalBefore, uint256 normalizedBefore) = broker.dynamicLoanPositions(borrower);
    uint256 rate = rateCalc.accrueRate(address(broker));
    uint256 actualDebt = BrokerMath.denormalizeBorrowAmount(normalizedBefore, rate);
    uint256 outstandingInterest = actualDebt > principalBefore ? actualDebt - principalBefore : 0;
    uint256 expectedNormalizedDelta = BrokerMath.normalizeBorrowAmount(actualDebt, rate);

    vm.prank(borrower);
    broker.convertDynamicToFixed(principalBefore, 52);

    (uint256 principalAfter, uint256 normalizedAfter) = broker.dynamicLoanPositions(borrower);
    assertEq(principalAfter, 0, "dynamic principal should be cleared");
    assertEq(normalizedAfter, 0, "dynamic normalized debt should be cleared");

    FixedLoanPosition[] memory fixedPositions = broker.userFixedPositions(borrower);
    assertEq(fixedPositions.length, 1);
    assertEq(
      fixedPositions[0].principal,
      principalBefore + outstandingInterest,
      "fixed principal should equal full outstanding debt"
    );

    // sanity: normalized delta consumed the whole normalized debt (allowing rounding wiggle)
    assertApproxEqAbs(expectedNormalizedDelta, normalizedBefore, 1, "normalized debt delta rounding");

    uint256 balanceBefore = IERC20(address(LISUSD)).balanceOf(borrower) ;
    IERC20(address(LISUSD)).approve(address(broker), type(uint256).max);
    vm.prank(borrower);
    broker.repay(borrowAmt, fixedPositions[0].posId, borrower);
    uint256 balanceAfter = IERC20(address(LISUSD)).balanceOf(borrower) ;
    assertEq(fixedPositions.length, 1, "Borrower should still have  a fixed position");
    vm.prank(borrower);
    moolah.withdrawCollateral(marketParams, COLLATERAL, borrower, borrower);  //this shouldn't be possible
    assertEq(balanceBefore - balanceAfter,borrowAmt, "borrower only payed the initial borrow amount");
  }
```

Recommendation: Consider adding the dynamic interest convert to principal at the broker also as principal at Moolah. This should ensure accounting to be in synch and enforce the proper repayment of the dynamic interest as principal for the fixed position.

Lista DAO: Fixed in commit 247299b.

Cantina Managed: Fixed.

3.2 Medium Risk

3.2.1 Missing `pause` and `unpause` functions in `LendingBroker.sol`

Severity: Medium Risk

Context: LendingBroker.sol#L33

Description: `LendingBroker` uses `whenNotPaused` guards but doesn't expose `pause()` or `unpause()` functions. The `PAUSER` role is granted but unused.

Recommendation: Consider implementing the `PAUSER` functions:

```
function pause() external onlyRole(PAUSER) {
      _pause();
  }

  function unpause() external onlyRole(MANAGER) {
      _unpause();
  }
```

Lista DAO: Fixed in commit 43021f34.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Moolah should check the broker's market Id when setting a new broker

Severity: Low Risk

Context: Moolah.sol#L261-L268

Description: When setting a new broker on Moolah `setMarketBroker()` only checks the broker's loan token and collateral token to be the same as those for the market Id being linked to it. However other parameters like the oracle used might be different and could lead to problems.

Recommendation: Query the broker for the market Id it has set and compare the Id it's being linked to.

Lista DAO: Fixed in 3409ab3

Cantina Managed: Fixed.

3.3.2 Amount to shares rounding can lead to small overpayments

Severity: Low Risk

Context: LendingBroker.sol#L822-L836

Summary: Rounding down of the amount conversion to shares can lead to users overpaying small amounts (e.g. less than the price of one share) on each repayment. If repayments are very frequent and share prices are high these amounts can add up.

Finding Description: When repaying principal amounts to Moolah the provided amount is converted to shares and rounded down.When the resulting number of share is less then the user's position the amount is provided rather than the shares.

In `Moolah:repay()` the amount is also rounded down when converting into shares. The rounded down shares is what is ultimately subtracted from the users position. This means that the amount above the rounded amount of shares is essentially forfeighted to Moolah.

When the user interacts with Moolah directly they have the opportunity to specify the shares to avoid this rounding effect. With the LendingBroker however they do not.

Impact Explanation: As the impact is limited to the price of one share per payment and payments are presumed to be orders of magnitude larger then the price of one share the impact is low.

Likelihood Explanation: as payments are subject to interest and penalties that change every block it is unlikely the amount will be exactly multiples of share prices.

Recommendation: Consider always specifying the number of shares when repaying Moolah.

Lista DAO: Fixed in commit 155db8f8.

Cantina Managed: Fixed.

3.3.3 Complete protocol DoS when rate accrual calculation overflows

Severity: Low Risk

Context: RateCalculator.sol#L154-L155

Description: The _rpow function in `BrokerMath.sol` performs exponentiation to calculate compounded interest rates. When the ratePerSecond is moderately high (e.g., `1.0007e27`) and sufficient time elapses between accruals (e.g., 86,400 seconds / 1 day), the intermediate multiplication operations within `_rpow` overflow, causing the function to revert with no error message.

When x (`ratePerSecond`) is `1.0007e27` and n (time elapsed) is 86,400 seconds, the repeated squaring operations cause xx to exceed `type(uint256).max`, triggering the overflow check. This failure cascades through the entire protocol as almost all critical operations depend on the rate calculator:

- `accrueRate()` - Cannot update rates.
- `getRate()` - Cannot retrieve current rates.
- `setRatePerSecond()` - Cannot update rate parameters (calls _accrueRate() internally).

Proof of Concept: The following proof of concept demonstrates how the User's repay got blocked due to overflows:

```
pragma solidity ^0.8.28;

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

/// local imports
import { Moolah } from "src/moolah/Moolah.sol";
import { ERC20Mock } from "src/moolah/mocks/ERC20Mock.sol";
import { OracleMock } from "src/moolah/mocks/OracleMock.sol";
import { IrmMock } from "src/moolah/mocks/IrmMock.sol";
import { MoolahVault } from "src/moolah-vault/MoolahVault.sol";
import { IMoolah, MarketParams, Id, Position, Market } from "moolah/interfaces/IMoolah.sol";
import { MarketParamsLib } from "moolah/libraries/MarketParamsLib.sol";
import { BrokerMath, RATE_SCALE } from "src/broker/libraries/BrokerMath.sol";

import { LendingBroker } from "src/broker/LendingBroker.sol";
import { RateCalculator } from "src/broker/RateCalculator.sol";

import "forge-std/console2.sol";

contract AuditTest is Test {
    using MarketParamsLib for MarketParams;

    /// roles
    address ADMIN = makeAddr("ADMIN");
    address MANAGER = makeAddr("MANAGER");
    address PAUSER = makeAddr("PAUSER");
    address BOT = makeAddr("BOT");
    address SUPPLIER = makeAddr("SUPPLIER");

    address BORROWER = makeAddr("BORROWER");

    ERC1967Proxy public moolahProxy;
    Moolah public moolah;
    MoolahVault public vault;
    LendingBroker public broker;
    RateCalculator public rateCalculator;

    /// mocks
    ERC20Mock public loanToken;
    ERC20Mock public collateralToken;
    OracleMock public oracle;
    IrmMock public irm;

    MarketParams public marketParams;
    Id public id;

    function setUp() external {
        _deployContracts();
        _configureContracts();
    }

    function test_brokerMath() external {
        BrokerMath._rpow(1.0007e27, 86400, 1e27);
    }

    function test_audit() external {
        _supplyCollateralToMoolah();
```

```
    vm.startPrank(BORROWER);
    broker.borrow(10e18);

    vm.warp(block.timestamp + 1 days);
    loanToken.setBalance(BORROWER, 100e18);
    loanToken.approve(address(broker), 100e18);

    broker.repay(25e18, BORROWER);
}

/// test helpers
function _supplyCollateralToMoolah() internal {
    collateralToken.setBalance(BORROWER, 100e18);
    vm.startPrank(BORROWER);
    collateralToken.approve(address(moolah), 100e18);
    moolah.supplyCollateral(marketParams, 100e18, BORROWER, bytes(""));
    vm.stopPrank();
}

/// internal helpers
function _deployContracts() internal {
    Moolah moolahImpl = new Moolah();

    moolahProxy = new ERC1967Proxy(
        address(moolahImpl),
        abi.encodeWithSelector(Moolah.initialize.selector, ADMIN, MANAGER, PAUSER, 0)
    );

    moolah = Moolah(address(moolahProxy));

    /// deploy tokens
    loanToken = new ERC20Mock();
    loanToken.setName("LOAN_TOKEN");
    loanToken.setSymbol("LOAN");
    loanToken.setDecimals(18);

    collateralToken = new ERC20Mock();
    collateralToken.setName("COLLATERAL_TOKEN");
    collateralToken.setSymbol("COLLATERAL");
    collateralToken.setDecimals(18);

    /// deploy oracle
    oracle = new OracleMock();
    oracle.setPrice(address(loanToken), 1e8);
    oracle.setPrice(address(collateralToken), 1e8);

    /// deploy irm
    irm = new IrmMock();

    vm.startPrank(MANAGER);
    moolah.enableIrm(address(irm));
    moolah.enableLltv(80 * 1e16); // 80%
    vm.stopPrank();

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

    RateCalculator rcImpl = new RateCalculator();
    ERC1967Proxy rcProxy = new ERC1967Proxy(
        address(rcImpl),
        abi.encodeWithSelector(
            RateCalculator.initialize.selector,
            ADMIN, MANAGER, PAUSER, BOT
            )
    );
    rateCalculator = RateCalculator(address(rcProxy));

    LendingBroker bImpl = new LendingBroker(address(moolah), address(vault), address(oracle));
    ERC1967Proxy bProxy = new ERC1967Proxy(
    address(bImpl),
    abi.encodeWithSelector(LendingBroker.initialize.selector, ADMIN, MANAGER, BOT, PAUSER,
    ↪  address(rateCalculator), 10)
    );
    broker = LendingBroker(payable(address(bProxy)));
}
```

```
    function _configureContracts() internal {
        /// create market
        marketParams = MarketParams({
            loanToken: address(loanToken),
            collateralToken: address(collateralToken),
            oracle: address(broker),
            irm: address(irm),
            lltv: 80 * 1e16
        });

        id = marketParams.id();
        moolah.createMarket(marketParams);

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

        rateCalculator.registerBroker(address(broker), 1.0007e27, 2e27);
        moolah.setMarketBroker(id, address(broker), true);
        broker.setFixedTermAndRate(
            1,
            1 days,
            2e27
        );
        vm.stopPrank();

        vm.startPrank(SUPPLIER);
        loanToken.setBalance(SUPPLIER, 100_000_000e18);
        loanToken.approve(address(moolah), type(uint256).max);
        moolah.supply(marketParams, 100_000_000e18, 0, SUPPLIER, bytes(""));
        vm.stopPrank();
    }
}
```

Recommendation: The issue happens only if the accrueRate function is not called for an extended period / if the rate per second is too high. Hence, consider implementing either of the following fixes:

1. Consider limiting the rate per second configuration to a safer limit, which will not overflow in the `_rpow` function even if called after a significant safe delay (~30 days).
2. Or consider changing the exponentiation function to a linear approximation to avoid the entire exponentiation that overflows.

If acknowledged, ensure that the off-chain accrual rate calling CRON is robust and doesn't have much redundancy.

Lista DAO: Acknowledged. Since the rate won't be high enough to trigger the issue, our bot will ensure the rate is accrued every hour, or even more frequently.

Cantina Managed: Acknowledged.

3.3.4 Centralization risk in `refinanceMaturedFixedPositions()`

Severity: Low Risk

Context: LendingBroker.sol#L728

Description: The `refinanceMaturedFixedPositions()` function can only be called by the BOT role. However, if a user's position matures and the bot is delayed or fails, the user has no way to trigger refinancing manually. The bot can also selectively censor users from refinancing their fixed-term positions, introducing centralization risk.

Recommendation: Consider allowing users to call this function on their own positions after a grace period, or document this limitation clearly.

Lista DAO: Acknowledged. Will let user know by putting this info at the documentation.

Cantina Managed: Acknowledged.

3.3.5 Missing pause functionality

Severity: Low Risk

Context: RateCalculator.sol#L12

Description: The `RateCalculator.sol` contract defines a PAUSER role but doesn't implement any pause functionality. This means if a critical issue is discovered, there's no way to halt rate updates temporarily.

Recommendation: Consider implementing the pause functionality (or) removing the PAUSER redundant role.

Lista DAO: Fixed in commit 43021f34 by removing the `PAUSER` role.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Remove unused file imports

Severity: Informational

Context: BrokerMath.sol#L4-L6, BrokerMath.sol#L9

Description: Importing unused files may decrease code quality and hinder maintenance over time. In the `BrokerMath.sol` file, imports including `IERC20`, `SafeERC20`, `IERC20Metadata`, `IBroker`, and `FixedTermAndRate` are redundant and remain unused.

Recommendation: Consider removing the above-mentioned unused file imports.

Lista DAO: Fixed in commit 23464ed4.

Cantina Managed: Fix verified.

3.4.2 Redundant state variable initialization in `LendingBroker` initializer

Severity: Informational

Context: LendingBroker.sol#L149-L150

Description: The state variables fixedPosUuid and borrowPaused are initialized to their default values in the `initialize` function, and are redundant. Default values are already zero and false for these variables; hence, explicit initialization is not required.

Recommendation: Consider removing the initializations of the above-mentioned state variables.

```
- fixedPosUuid = 0;
- borrowPaused = false;
```

Lista DAO: Fixed in commit fce33ed3.

Cantina Managed: Fix verified.

3.4.3 `DOMAIN_SEPARATOR` immutable variable uses wrong address

Severity: Informational

Context: Moolah.sol#L97

Description: In `Moolah.sol`, the DOMAIN_SEPARATOR is an immutable variable computed in the constructor using `address(this)`. In the UUPS proxy pattern, constructors execute within the context of the implementation contract, not within the context of the proxy itself. This means that DOMAIN_SEPARATOR is computed using the implementation address rather than the proxy address, which breaks EIP-712 signature verification for the `setAuthorizationWithSig()` function.

```
// Moolah.sol constructor
constructor() {
    _disableInitializers();
    DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, block.chainid, address(this)));
    // address(this) = implementation address, NOT proxy address
}
```

Recommendation: Convert DOMAIN_SEPARATOR from an immutable to a view function that computes the value using the current contract address (proxy):

```
- bytes32 public immutable DOMAIN_SEPARATOR;

  constructor() {
      _disableInitializers();
-     DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, block.chainid, address(this)));
  }

+ function DOMAIN_SEPARATOR() public view returns (bytes32) {
+     return keccak256(abi.encode(DOMAIN_TYPEHASH, block.chainid, address(this)));
+ }
```

Alternatively, compute in the initializer and store as a regular state variable (saves GAS):

```
bytes32 public DOMAIN_SEPARATOR;

function initialize(...) public initializer {
    // ... existing initialization ...
    DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, block.chainid, address(this)));
}
```

> Note: The team also identified this issue during the audit, so it is classified as informational. Furthermore, this change is outside the scope of this audit, and the issue was created for documentation purposes.

Lista DAO: Fixed in commit 4f2e4c1b.

Cantina Managed: Fix verified.

3.4.4 Loan token allowance from broker to Moolah is not reset when not used

Severity: Informational

Context: LendingBroker.sol#L817-L837

Description: Whenever there repaying Moolah the loan token allowance is increased. If the full amount is not used the funds are refunded to the user but the allowance is not reduced. This will lead to an ever increasing allowance for Moolah. Although the broker is not supposed to hold funds at rest and Moolah currently has no way of using this allowance inappropriately it is better to reduce the allowance when not needed.

Recommendation: Reduce the allowance with the same amount as used for refunding the user.

Lista DAO: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 `repayPrincipalAmt` variable shadowing in `_deductFixedPositionsDebt()`

Severity: Informational

Context: LendingBroker.sol#L559-L572

Description: `repayPrincipalAmt` is defined on both L559 and L572 causing shadowing. The one on L572 is never used.

Recommendation: Remove the `repayPrincipalAmt` on L572 to avoid confusion.

Lista DAO: Fixed in commit 247299b7.

Cantina Managed: Fix verified.

3.4.6 User repaying a fixed position early is paying a penalty on the penalty itself

Severity: Informational

Context: LendingBroker.sol#L315-L319

Description: When a user repays a fixed position before the deadline they are charged a penalty based on the amount repayed and the remaining time. This implementation however calculates the penatly as `amount * apr * time` and deducts that penalty from the amount; Only the remaining amount is used to repay the position. Technically this means the user is paying a penalty on the penalty itself.

Recommendation: Although the formula for the penalty is ultimately a design decision, consider using a formula that only applies the penalty to the amount used for the repayment (e.g. `amount * PenaltyPercentage /(PenaltyPercentage + 1)`).

Lista DAO: Acknowledged.

Cantina Managed: Acknowledged.

3.4.7 Incompatible use of immutable variables in UUPS upgradeable contract

Severity: Informational

Context: LendingBroker.sol#L48-L50

Description: The `LendingBroker.sol` contract implements the UUPS (Universal Upgradeable Proxy Standard) pattern by inheriting from UUPSUpgradeable. However, it declares three critical state variables as immutable:

```
IMoolah public immutable MOOLAH;
IMoolahVault public immutable MOOLAH_VAULT;
IOracle public immutable ORACLE;
```

This design pattern is problematic for upgradeable contracts because immutable variables are stored in bytecode, not storage. When the contract is upgraded to a new implementation, the proxy delegates calls to the new implementation contract. The new implementation will have its own immutable variables set at its deployment time, which may differ from the original implementation.

Recommendation: Convert all immutable variables to regular storage variables and initialize them in the initialize function.

Lista DAO: Acknowledged.

Cantina Managed: Acknowledged.

3.4.8 Increase test coverage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: There is less than complete test coverage of key contracts under review. Adequate test coverage and regular reporting are essential processes in ensuring the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions arising due to changes in the underlying smart contract implementation.

Recommendation: Add to test coverage to ensure all execution paths are covered. It is highly recommended to do fuzz and invariant testing for LendingBroker and BrokerMath as these contracts are math heavy and edge cases could only be identified through such tests.

Lista DAO: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 Use consistent naming convention

Severity: Informational

Context: LendingBroker.sol#L644

Description: The variable _position uses an underscore prefix, which is typically reserved for internal/private functions or to avoid shadowing. This naming pattern is inconsistent with the rest of the codebase.

```
Position memory _position = MOOLAH.position(MARKET_ID, user);
if (_position.collateral == 0) {
```

Recommendation: Use consistent naming conventions. Consider renaming to userPosition or position.

Lista DAO: Acknowledged.

Cantina Managed: Acknowledged. The code is now moved to `BrokerMath.sol` (without fixing).

3.4.10 Replace magic numbers with constants

Severity: Informational

Context: LendingBroker.sol#L624, BrokerMath.sol#L83, BrokerMath.sol#L95

Description:

1. The `LendingBroker.sol` contract uses magic numbers like 10 8** for returning the loan token price without defining them as constants:

```
return 10 ** 8;
```

2. The `BrokerMath.sol` library uses magic numbers like 365 days and 2:

```
return Math.mulDiv(apr - RATE_SCALE, 1, 365 days, Math.Rounding.Floor);
// ...
penalty = Math.mulDiv(
  Math.mulDiv(repayAmt, _aprPerSecond(position.apr), RATE_SCALE, Math.Rounding.Ceil), // repayAmt * APR(per
  ↪  second)
  timeLeft,
  2,
  Math.Rounding.Ceil
);
```

Recommendation: Define magic numbers as named constants:

```
+ uint256 private constant LOAN_TOKEN_PRICE = 10 ** 8;

  if (token == LOAN_TOKEN) {
-   return 10 ** 8;
+   return LOAN_TOKEN_PRICE;
  }


  /// BrokerMath.sol
+ uint256 private constant PENALTY_DIVISOR = 2;
+ uint256 private constant SECONDS_PER_YEAR = 365 days;
```

Lista DAO: Fixed in commit 0928f507.

Cantina Managed: Fix verified.

3.4.11 Redundant return in `setRatePerSecond()` function

Severity: Informational

Context: RateCalculator.sol#L108

Description: The `setRatePerSecond()` function has a return statement for a void internal function:

```
function setRatePerSecond(address _broker, uint256 _ratePerSecond) external onlyRole(BOT) {
 return _setRatePerSecond(_broker, _ratePerSecond);
}
```

The internal function `_setRatePerSecond()` doesn't return any value, making the return keyword unnecessary.

Recommendation: Consider removing the return keyword:

```
  function setRatePerSecond(address _broker, uint256 _ratePerSecond) external onlyRole(BOT) {
-   return _setRatePerSecond(_broker, _ratePerSecond);
+   _setRatePerSecond(_broker, _ratePerSecond);
  }
```

Lista DAO: Fixed in commit 44c77129.

Cantina Managed: Fix verified.

3.4.12 Declare all `BrokerMath` and `PriceLib` library functions internal

Severity: Informational

Context: BrokerMath.sol#L13, PriceLib.sol#L9

Description: Several library functions are declared as public when they could be internal. This increases deployment costs and gas usage. The following functions are marked public:

- `GetTotalDebt()`.
- `MulDivCeiling()`.
- `MulDivFlooring()`.
- `GetAccruedInterestForFixedPosition()`.
- `GetPenaltyForFixedPosition()`.
- `PreviewRepayFixedLoanPosition()`.
- `NormalizeBorrowAmount()`.
- `DenormalizeBorrowAmount()`.
- `_rpow()`.
- `_rmul()`.
- `_getPrice()`.

Recommendation: Change all library functions to internal unless they need to be called externally.

> Note: If these are intentionally public for external contract calls, document this decision.

Lista DAO: Acknowledged. As changing those functions to internal will embed them into the runtime byte code which causes the runtime contract size exceeding the limit.

Cantina Managed: Acknowledged.

3.4.13 Fix typos

Severity: Informational

Context: PriceLib.sol#L34

Description: Fix the following typos identified in the codebase under the scope of the audit:

```
- // price deviatiates with user's position at broker
+ // price deviates with user's position at broker
```

Lista DAO: Fixed in commit e44f7dff.

Cantina Managed: Fix verified.