Lista Dao
Fixed Term Lending
Follow-up Audit

FINAL REPORT

October ‘2025

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Lista Dao – Fixed Term Lending – Follow-up Audit Report
Website	lista.org
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/lista-dao/moolah/tree/63389282bf87054376b4e103932d118326f7d023
Resolution 1	

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no changes made)	Failed resolution	Open
High	11					
Medium	5					
Low	9					
Informational	7					
Governance	1					
Total	33					

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

LendingBroker

This review represents a follow-up audit of the scope referenced at the beginning of this document. Due to major refactors and design changes arising from vulnerability mitigations post the initial audit, a complete re-audit of the scope was recommended to the Lista DAO team to ensure greater security confidence.

The LendingBroker contract is the main contract in the protocol that handles the loans. There are 2 types of loans offered by the system: dynamic loans, with dynamic interest rates, and fixed-rate loans with fixed rates and loan durations.

The contract interacts with an exclusive Moolah vault, where only the broker is allowed to borrow from and repay to. This special vault is also set up with 0 interest rates, and the actual interest calculations are handled by the LendingBroker instead.

When a user borrows funds, the contract borrows from the Moolah vault on the user's behalf and thus requires to be whitelisted by the user on Moolah. During repayments, the interest is deducted and kept by the broker, while the principal is repaid back to the Moolah vault under the same user's account.

Liquidations are carried out by calling the liquidate function. The current Liquidator and PublicLiquidator contracts do not yet support this functionality since it is still under development and out of scope for this audit.

Appendix: Dynamic loans

Dynamic loans apply a variable interest rate to the position. When users borrow funds, their debt is denominated in shares according to the current getRate value. The rate then keeps increasing over time, requiring higher repayments to nullify the same borrowed shares.

The interest accrual is handled in the RateCalculator contract. The Bot sets a rate per second via the setRatePerSecond function, and the getRate value keeps increasing based on this value. The accrueRate function updates the current interest state of the contract.

During repayments, priority is given to the interest amount, and only after clearing the interest can the actual principal be repaid.

Appendix: Fixed-term loans

Fixed-term loans are loans with a constant interest rate for a constant loan duration. Unlike dynamic loans, where the interest rate can be changed by the Bot anytime, fixed-term loans record the interest rate at the very beginning and use simple interest calculation to calculate the interest owed based on that amount.

Users can own multiple fixed-rate loan positions up to a specified maxFixedLoanPositions limit. Whenever repayments are made, the contract records the repaid interest and repaid principal in separate variables. If a user repays the loan before the end of the term, they attract an extra penalty which is equal to half of the remaining interest.

Appendix: Interest handling

Interest is handled differently for the different loan types.

For dynamic loans, the loans are denominated in borrow shares according to the current interest rate value, which keeps going up over time. During repayment, the shares to be paid back are calculated according to the interest rate value at that moment. The contract deducts the borrow shares based on the repaid value and also keeps track of the principal amount separately. When the entire principal is repaid, the position is deleted.

For fixed-term loans, the interest is calculated using the fixed interest rate. Interest repayments are recorded in an interestRepaid variable, and if any of the principal is also repaid, a penalty is added on top.

Appendix: Loan conversions

Dynamic loans can be converted to fixed-rate loans via the convertDynamicToFixed function. This nullifies the user's borrow shares and instead opens a fresh fixed-term loan of the principal + interest amount.

Appendix: Liquidation

The liquidation mechanism has been heavily modified to accommodate the Broker contract charging interest instead of Moolah.

Firstly, the Broker implements an oracle for the collateral token. Unlike normal Moolah vaults, where the collateral is priced with an oracle directly, here the collateral price is actually scaled down based on the accrued interest amount according to the following formula.

> collateralPrice = oraclePrice - AccruedDebt/TotalCollateral

By scaling down the collateral price, the Moolah vault can now effectively compare the collateral with the loan, including the interest.

Once a position is liquidatable, whitelisted liquidators can call the contract and pay off part of the principal and accrued interest and seize the user's collateral tokens. After liquidation, the dynamic and fixed-term positions are adjusted to reflect the new liquidated state. The dynamic position is adjusted first, repaying its collateral and interest. Then the fixed-term positions are adjusted one by one, from the loan with the earliest end time to the latest.

Appendix: Economic simulator

A simulator was set up to see the behaviour of the system on Desmos. The initial conditions chosen were of both stablecoins as collateral and loan, and an lltv of 0.8 for the market, which leads to an incentive factor of 1.0638. The graph can be found here. This was used to investigate a number of issues.

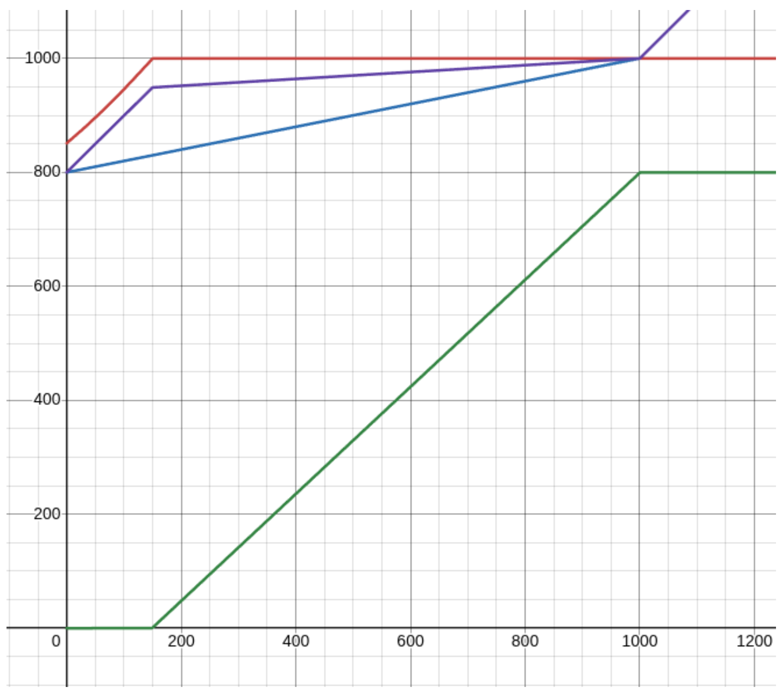
In the graph,

- The X-axis is the amount of interest accrued.
- p_collat is the price of collateral. Given by (1-x/x_collat), capped to 0.
- i_incentive is the incentive factor of the market
- L_loan is the loan amount. This can be adjusted to check various scenarios.
- D_debtMax (Blue line) is the total debt to the system. This includes the Moolah borrow (x_collat x p_collat x 0.8) and the interest (x).
- A_seized (Red line) is the total amount seized on a liquidation. This is given to the liquidator.
- A_paid (Purple line) is the total amount paid by the liquidator, including interest. This scenario always goes for the maximum liquidation amount, including bad debt scenarios.
- D_badDebt (Green line) is the amount of bad debt incurred in the system.

The mathematical expressions of the various quantities can be found in the hyperlink.

The following is a snapshot when there's a loan of 800 tokens.

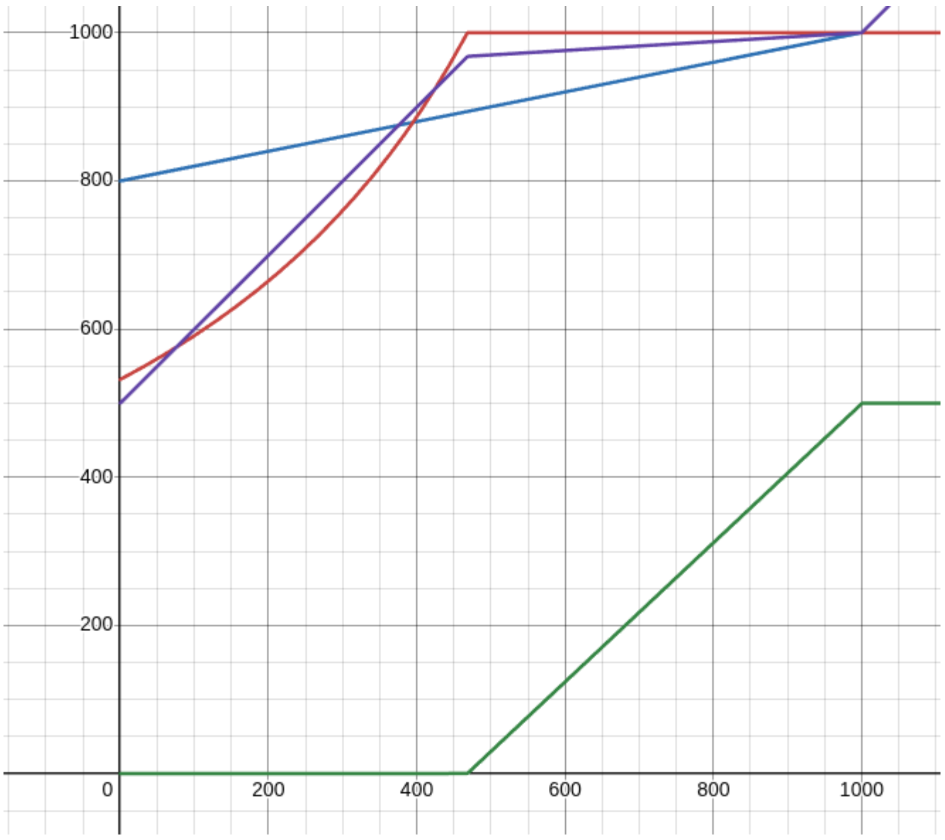
藍: total debt
紅: max seized
紫: repaid ASSETS
綠: bad debt



The Blue line (D_debtMax) increases as the X-axis progresses (interest). This shows that even though Moolah only allows an lltv of 0.8, since here you can have debt as interest, the system has an lltv which increases as the interest accrues. Liquidations can occur at any point above this blue line, not below it. The Red line A_seized is the amount of collateral assets seized. This reaches the maximum collateral of 1000 and then stays fixed. The purple line A_paid is the amount of principal+interest paid by the liquidator. If this line is below the red line (siezed > paid), it means liquidations are profitable. The green line shows the bad debt, which only starts after the entire collateral is exhausted.

藍: total debt
紅: max seized
紫: repaid ASSETS
綠: bad debt

Some issues can be identified by changing the loan size to 500.



Here, one can observe a triangular slice bounded by the blue, purple, and red lines around the point (400,900). This is above the blue line, thus liquidation is possible here. But the red line is below the purple line, so more needs to be paid to do the liquidation than is seized, and is thus unprofitable. Also note how, as the loan amount is decreased, the first possible liquidation point (where the purple line goes above the blue line) shifts to the right. For a loan of 1 token, this intersection is very close to 1000 interest, forcing bad debt very easily onto the system.

Privileged Functions

- onMoolahLiquidate
- refinanceMaturedFixedPositions
- setMarketId
- updateFixedTermAndRate
- setMaxFixedLoanPositions
- setBorrowPaused
- pause
- unpause
- toggleLiquidationWhitelist

Core Invariants:

INV 1: Sum of loan principals must reflect the total amount borrowed from Moolah.
INV 2: During repayments, the interest amount is always repaid before the principal.
INV 3: Each loan position must be of at least minLoan value

Issue_01	LLTV can be violated due to interest accrual
Severity	**High**
Description	The Moolah vault enforces an LLTV (say 80%) on its positions. However, the overall LLTV of the loan, including the accrued interest, can be much higher, depending on the size of the loan. This is due to how the liquidation situation is calculated. The maximum borrow amount from Moolah is defined as: > collateral x price_collateral x lltv However, since there is interest included as well, the actual total borrow amount, including this loan, is defined as: > interest + collateral x price x lltv Assume we only work with stablecoins, and there are 1000 tokens collateral and a loan of 500 tokens. On moolah, an lltv of 0.8 would mean that the max borrow is 800 tokens. So the position can accrue 300 tokens of interest before it becomes liquidatable. But here, even if 301 interest accrues, due to how the collateral price is scaled, the position will still not be liquidatable. debtAtMoolah = 500 debtAtBroker = 801 (above lltv! should be liquidatable) deltaDebt = 301 deduction = 0.301 collateral price = 1-0.301 = 0.699 Now the max borrow is = 1000 x 0.699 x 0.8 = 559.2 But since the moolah loan is only 500 tokens, the position is still safe. Thus, the LLTV of the system is actually higher than that of Moolah, and it actually fluctuates with the interest amount. This violates the risk parameters of the protocol.

	This issue is also described graphically in Appendix: Simulator.
Recommendations	Scaling down the collateral price is causing this issue. Re-think the liquidation logic without scaling down the collateral price, and instead incorporate the loan into the debt in Moolah.
Comments / Resolution	

Issue_02	Protocol can be forced into bad debt
Severity	**High**
Description	Due to the scaling down of the collateral price, the lltv of the system increases as explained in the issue above. In the extreme case, a very small loan can reach a situation where it turns into bad debt before it can be liquidated. Let us consider a situation where there are 1000 tokens collateral, and 1 token loan in a 0.8 lltv market. Now, say the position accrues 998 tokens of interest. debtAtMoolah = 1 debtAtBroker = 999 deltaDebt = 998 deduction = 0.998 collateral price = 1-0.998 = 0.002 max borrow = 1000 x 0.002 x 0.8 = 1.6. Since the debt at moolah is 1<1.6, this position is still not liquidatable. In fact, this position only becomes liquidatable once the accrued interest hits 998.75 tokens. But the moment the interest hits 998.937, the position turns into bad debt. This means that if interest is higher than that, all 1000 of the collateral will be seized, but less than the required amount of loan tokens will be paid off. So the position has to be liquidated in the very small window between 998.75 - 998.937 loan tokens. Any failure will lead to bad debt. If the loan accrues to 999, debtAtMoolah = 1 debtAtBroker = 1000 deltaDebt = 999 deduction = 0.999 collateral price = 0.001

	incentivefactor = 1.0638 (based on 0.8 lltv) So all the collateral will seized, and the amount repaid will be: 1000 x 0.001 / 1.0638 = 0.94 So 0.94 tokens will be paid back, and the rest 0.06 tokens will be bad debt. These numbers can be scaled up to ensure large amounts of bad debt accrued in the system.
Recommendations	Re-work the collateral pricing method. The collateral price decrements prevent loan liquidations by changing the net lltv of the protocol, preventing liquidations.
Comments / Resolution	

Issue_03	Regions of unprofitable liquidations
Severity	**High**
Description	Normally, in Moolah vaults, liquidations are always profitable, thus always incentivized. Even if there isnt enough collateral, only a part of the debt needs to be paid back to seize the entire collateral and the rest becomes bad debt. This is important to ensure liquidators are active, protecting the health of the protocol. However, here, due to how the collateral price is scaled, there are pockets where liquidations are straight-up unprofitable. For example, consider the following situation: Collateral of 1000 tokens, loan = 500 tokens, lltv = 0.8 Interest accrues to 374 tokens. deltaDebt = 374, collateral price = 1-0.374 = 0.626 max allowed debt = 1000 x 0.626 x 0.8 = 500.8. The loan isn't liquidatable yet, since the moolah debt (500) is less than the max 500.8. If the interest accrues to 376 tokens, it becomes liquidatable. deltaDebt = 376, collateral price = 1-0.376 = 0.624 max allowed debt = 1000 x 0.624 x 0.8 = 499.2. Since 500 > 499.2, this can be liquidated. For liquidation, the incentive factor is 1.0638 for 0.8 lltv. So the user pays off the debt of 500 tokens, and receives collateral of 500x1.0638/0.624=852.4 tokens. However, the liquidator also needs to pay off the 376 tokens of interest, thus a total of 876 tokens, while they get only 852.4 tokens of collateral. Thus, this is unprofitable. This becomes profitable only when interest accrues to 425 tokens. Then,

	deltaDebt = 425, collateral price = 1-0.425 = 0.575 paying off 500 loan gives 500x1.0638/0.575 = 925.04 tokens. This is used to pay off the 500 loan + 425 interest with a 0.4 profit. Thus, liquidations are not always profitable. This is also true if the interest ever accrues to be more than the collateral itself, in which case the system prioritizes paying off the interest first, so liquidations are always unprofitable then. Profitable liquidations are necessary to keep the system healthy. This issue is also described graphically in Appendix: Simulator.
Recommendations	Re-work the liquidation method. The broker should forego interest to keep the liquidations profitable if required.
Comments / Resolution	

Issue_04	Liquidations permanently fail due to access control check in Liquidator.onMoolahLiquidate
Severity	**High**
Description	During the liquidation flow, the onMoolahLiquidate() function in the LendingBroker contract calls the onMoolahLiquidate() function in the Liquidator, which has the check msg.sender == MOOLAH. Since the msg.sender is the broker and not moolah, the call reverts and liquidations fail. *require(msg.sender == MOOLAH, OnlyMoolah());*
Recommendations	Consider implementing a liquidator that allows the call to succeed while ensuring enough loan tokens are approved to the broker to complete repayment.
Comments / Resolution	

Issue_05	User can be DOSd with partial liquidations
Severity	High
Description	The contract has a check on the repay function, _validatePositions, which makes sure that every position owned by the user is above a minimum amount; otherwise, it reverts. If a user has 2 positions below the minimum amount, they will be locked from their account since trying to repay one of the loans will revert due to the presence of the other loan. This situation can actually be triggered by partial liquidations. Assume a user has 3 positions: 1. dynamic loan of 100 principal + 10 interest 2. Fixed loan of 100 principal + 0 interest 3. Fixed loan of 400 principal + 10 interest Thus, totalDebtAtBroker = 620 debtAtMoolah=600 If they are liquidated with an amount of 100, interestToBroker=100*620/600-100=3.333 So 100 principal and 3.33 interest are repaid. So the dynamic position becomes 0 principal, 6.67 interest. Fixed positions stay the same. Now, totalDebtAtBroker = 516.67 debtAtMoolah=500 If they are liquidated again with 80 amount, interestToBroker=80*516.67/500-80=2.667 So 80 principal and 2.667 interest is repaid. The dynamic position absorbs the interest repayment, becoming 0 principal, 0.66 interest. (below min loan)

	Position 2 absorbs the principal repayment, becoming 20 principal, 0 interest. If minLoan is above 20 tokens, then there are 2 loans (position 1 and 2) which are below this limit, leading to a lockout.
Recommendations	Consider enforcing minLoan checks after a liquidation occurs. Otherwise, overhaul the _validatePositions check.
Comments / Resolution	

Issue_06	Liquidations allow for unpaid interest to be zeroed out
Severity	**High**
Description	This occurs because in the function onMoolahLiquidate, after deducting from the dynamic position, any leftover value to be deducted is split into interestLeftover and principalLeftover. Next, the function will use these values to deduct value from the fixed positions. In the case where principalLeftover is enough to deduct all the principal of a fixed position, but not enough to zero out/eliminate all the interest, then the position will erroneously be deleted and the interest due with it.
Recommendations	The fix is non-trivial, consider ensuring that the interest is paid off before deleting a position.
Comments / Resolution	

Issue_07	Principal of a dynamic position getting set to 0 even when interest is non-zero allows the user to bypass interest repayment
Severity	**High**
Description	During liquidation, the principal of dynamic position can drop to 0 even when there is non-zero interest remaining in case (principalToDeduct >= dynamicPosition's principal && interestToDeduct < dynamicPosition's interest) This will allow the user to bypass the repayment of this interest since the bounded underlying collateral that necessitated repayment can now be moved by repaying the principals of fixed positions. For example: *collateral in moolah = 100* *debt in moolah = 80* *dynamic debt in broker = 40* *fixed debt in broker = 40* *total debt in broker = 80* *total interest in broker = 8* *interest in dynamic position = 6* *interest in fixed position = 2* now if 40 (ie. 50%) is liquidated, *interestToDeduct = 4* *principalToDeduct = 40* *dynamic position's principal = 40 - 40 = 0* *dynamic position's interest = 6 - 4 = 2* If the user repays the fixed position then debtInMoolah will become 0 and hence the user can withdraw their collateral freely without

	ever repaying the dynamic position’s interest of 2.
Recommendations	Re-work the liquidation mechanism, proportional reduction of interest and principal is flawed.
Comments / Resolution	

Issue_08	Incorrect values defined for MAX_FIXED_TERM_APR and MIN_FIXED_TERM_APR lead to inflated interest
Severity	**High**

Description	The LendingBroker contract defines the variables MAX_FIXED_TERM_APR and MIN_FIXED_TERM_APR as 31e27 and 2e27. However, they do not represent 30% and 2% but rather 3000% and 100%.
	
	*uint256 public constant MAX_FIXED_TERM_APR = 31 * RATE_SCALE; // 30%*
	*uint256 public constant MIN_FIXED_TERM_APR = 2 * RATE_SCALE; // 1%*
	
	Let's take an example to understand this:
	- Assume fixed term APR = 2e27.
	- In function getAccruedInterestForFixedPosition, assume principal - principalRepaid = 100e18 and timeElapsed = 365 days.
	- In _aprPerSecond, the following calculation occurs: 2e27 - 1e27 / 365 days = 1e27 / 365 days
	- When this value is returned to getAccruedInterestForFixedPosition, the calculation occurs as follows: (100e18 * (1e27/365 days) * 365 days) / 1e27 = 100e18 * 1e27 / 1e27 = 100e18.
	
	As we can see, over the course of 365 days, the user will be charged 100% of their principal as interest instead of 1%.

Recommendations	Consider updating the MAX_FIXED_TERM_APR and MIN_FIXED_TERM_APR variables to 1.01e27 and 1.3e27, respectively.
Comments / Resolution	

Issue_09	Bot can force bad debt and liquidations
Severity	**High**
Description	When the Bot calls refinanceMaturedFixedPositions to convert fixed positions to dynamic, there is no check to ensure that there are no duplicates in the posIds array. Thus, a malicious bot can actually include the same loan multiple times, and this will include the same debt multiple times in that user's position. This can be abused to make that position instantly liquidatable. Furthermore, since debt is accrued only on the broker but not on Moolak, there's no way to even repay this debt. If it is attempted, the moolah.repay call will fail, since this extra debt was never taken out of Moolah in the first place. This is a severe escalation of privilege and is thus a major issue.
Recommendations	Check for duplicates in the posIds array.
Comments / Resolution	

Issue_10	Broker prioritizes interest collection over bad debt prevention
Severity	**Medium**
Description	The protocol always makes sure the loan interest is paid in full to the broker. The issue is that in certain scenarios, this is done at the cost of accruing bad debt in the moolah vault. The moolah vault does not put any restrictions on the suppliers, so anyone can supply loan tokens to the vault. However, this means they will face bad debt in case of liquidations, while the broker always gets paid. As an example, say a position has 1000 tokens collateral and 800 tokens loan, with lltv of 0.8. Say it accrues 200 tokens in interest. deltaDebt = 200, collateral price = 1-200/1000=0.8 The liquidator seizes all 1000 collateral tokens. They pay off a part of the loan = 1000 x 0.8 / 1.0638 = 752 tokens. So in total, the liquidator pays 752 tokens to Moolah and 200 tokens in interest to the broker. They seize 1000 tokens and thus make a profit of 48 tokens. Thus the Broker gets paid 200 tokens of interest, but Moolah only gets paid 752 tokens of the 800 token loan. So the remaining 48 tokens loan get turns into bad debt and socialized. The Broker could have given up on some amount of interest to prevent bad debt in Moolah, but didn't.
Recommendations	Prioritize preventing bad debt in Moolah over collecting interest, since Moolah allows anyone to supply loan tokens, not just the relayer contract.
Comments / Resolution	

Issue_11	Incorrect inclusion of broker-level interest in checkPositionsMeetsMinLoan
Severity	**Medium**

Description	Function checkPositionsMeetsMinLoan ensures that all Moolah operations that occur through the broker respect the minLoan requirement.
	
	However, checkPositionsMeetsMinLoan currently takes into account the interest in the broker when this interest is not accounted for in moolah as the debt at moolah.
	
	uint256 dynamicDebt = denormalizeBorrowAmount(dynamicPosition.normalizedDebt, currentRate);
	if (dynamicDebt > 0 && dynamicDebt < minLoan) {
	    isValid = false;
	}
	
	uint256 fixedPosDebt = _fixedPos.principal -
	      _fixedPos.principalRepaid +
	      getAccruedInterestForFixedPosition(_fixedPos) -
	      _fixedPos.interestRepaid;
	    if (fixedPosDebt > 0 && fixedPosDebt < minLoan) {
	      isValid = false;
	}
	
	Due to this, the comparison is erroneous as it assumes the debt at moolah is higher than intended.

Recommendations	Consider not including interest when comparing against the minLoan.
Comments / Resolution	

Issue_12	Liquidations/repayments concerns during paused state
Severity	**Medium**
Description	When the broker is paused, interest continues to accrue on all positions. Since liquidations and repayments are also blocked during this state, it can result in protocol bad debt and previously healthy positions may become liquidatable by the time the market is unpaused. This can cause a sudden wave of unexpected liquidations immediately after unpausing.
Recommendations	Consider at least allow repayments and liquidations during paused state.
Comments / Resolution	

Issue_13	Converting dynamic positions to fixed reduces net health
Severity	**Medium**
Description	Due to interest accrual, the net lltv allowed by the system until liquidation changes, as described in a separate issue. A dynamic position has interest and a principal. When it is converted to a fixed-term position, the accrued interest is added to the principal. Thus, the net interest goes down, reducing the allowed lltv. For example, let's look at a situation where we have 1000 collateral tokens and 500 loan tokens, with lltv 0.8. Say the position has accrued an interest of 299 tokens. deltaDebt = 299, collateral price = 1-0.299=0.701 max borrow = 1000 x 0.701 x 0.8 = 560.8 Since the actual Moolah debt (500) is a lot lower than the max borrow (560.8), this position is pretty safe from being liquidated. In fact, this position is liquidatable only when the interest rises to 375. The 76 tokens of interest can still accrue safely. However, if we convert the same into a fixed-term loan, the interest gets added in as the loan principal. debtToMoolah = 799, debtToBroker = 799, deltaDebt = 0, collateral price = 1 max borrow = 1000 x 0.8=800. This is extremely close to the liquidation threshold. In fact this position is liquidatable the moment the interest rises to 1.25. So the position will be liquidiatble in a very short time. Thus the health factor, or duration until liquidation changed drastically just by changing the loan type. The root cause of this is that the effective lltv of the system is not constant and depends on the interest amount. Similarly, certain dynamic loans cannot be converted to fixed loans at all. In the previous example, if the 500 loan position accrued 301

	tokens of interest, it cannot be converted to a fixed-term loan, since Moolah will not allow a borrow of 801 tokens against a collateral of 1000 tokens. Thus there is a gap between what Moolah allows, and where the actual liquidation point lies.
Recommendations	Scaling down the collateral price is causing this issue. Re-think the liquidation logic without scaling down the collateral price, and instead incorporate the loan into the debt in Moolah.
Comments / Resolution	

Issue_14	Stale market conditions in getDebtAtMoolah and _repayToMoolah
Severity	Low
Description	It is stated that interest will not accrue in Moolah for markets that have an associated broker. However, there is nothing in the code that enforces this. If interest does accrue on Moolah, several functions that rely on up-to-date market data may operate with stale values. getDebtAtMoolah: This function is used exclusively during liquidations. If the liquidator hasn't interacted with Moolah before hitting the broker, the function will compute debt using stale values. This stale debt is then used in onMoolahLiquidate to compute the broker's interest share. If Moolah has actually accrued interest, the liquidator may end up paying more interest than expected. _repayToMoolah: This function is used for both fixed and dynamic repayments. Interaction with Moolah only occurs when _supplyToMoolahVault is called with accrued interest above minLoan. If there is no interest, or only minimal interest below minLoan, the relayer will not perform a supply operation to Moolah on behalf of the vault. As a result, stale market data is used to compute the repayment shares. This can cause repayments to fail, since Moolah will attempt to pull more assets than the allowance covers, effectively DoSing the repayments.
Recommendations	Consider calling Moolah::accrueInterest before interacting with the market data. Alternatively, it can also be acknowledged that theoretically, no interest will be accrued on Moolah markets when a broker is set.
Comments / Resolution	

Issue_15	Users can avoid penalty via self-liquidations
Severity	Low
Description	While a penalty is charged when closing fixed-term loans before the expiry, no such penalty is placed during liquidations. Thus, users can self-liquidate their positions to unwind loans without paying a penalty before the expiry time. One way to enable this would be to borrow the maximum possible amount and then wait for interest to accrue, forcing the position to become liquidatable.
Recommendations	Consider acknowledging this issue. Otherwise, consider placing penalty on fixed-term liquidations as well, however that can lead to unprofitable liquidations.
Comments / Resolution	

Issue_16	Liquidations are heavily dependent on liquidator contract logic
Severity	Low
Description	Currently the liquidation logic relies on correct execution of the liquidator contract and the liquidator contract relies on correct swap prices provided by the pool. This creates a heavy dependency on behaviours in the liquidator contract and pools. For example, if a pool is paused, liquidations for the broker will be paused as well.
Recommendations	It is recommended to monitor which liquidators are whitelisted and their underlying behaviour.
Comments / Resolution	

Issue_17	Liquidation misbehavior when callback data is not provided
Severity	**Low**
Description	Within the broker, onMoolahLiquidate must be called during liquidation. This callback is responsible for: - Approving Moolah to pull repaidAssets - Sending collateral to the liquidator - Clearing the liquidated position's debt and interest of the broker. If the liquidator provides empty callback data, onMoolahLiquidate will not be invoked. In most situations this should cause the liquidation to revert due to missing allowances, and this is indeed the behavior in the majority of cases. However, an alternative attack path exists: - The attacker donates repaidAssets to the broker via a normal transfer. - Performs a liquidation with empty data, preventing the callback from executing. - Because _repayToMoolah can leave small residual allowances for Moolah, these tiny allowances allow Moolah to pull the repaidAssets during liquidation even without calling the callback. This results in a state where the position is liquidated in Moolah, but the corresponding broker position state is not updated/cleared, leaving the position inconsistent. There is an issue with this scenario: the collateral extracted from the position becomes locked inside the broker contract. Because of this, the attacker has no direct incentive to perform the attack. Although we did not find any way to withdraw the locked collateral, an attacker could still execute the attack with a minimal amount, minimizing their loss. A possible consequence is that the liquidated position may not be fully closable in the broker, since part of the repayment was accounted for in Moolah but not properly deducted from the

	broker's position, enforcing that position to always accrue interest even if the owner wants to fully close it.
Recommendations	Enforce that liquidation callback data must be non-empty to guarantee that onMoolahLiquidate is always executed.
Comments / Resolution	

Issue_18	Race condition when changing fixed term configuration
Severity	**Low**
Description	A race condition may occur when a transaction updating a fixed term's APR and/or duration is pending in the mempool. If this update is executed before a user's transaction that selected the fixed term using the old configuration, the user's position will instead be created using the new parameters. As a result, the user may end up paying more interest and/or being locked for a longer duration than expected.
Recommendations	This behavior can be acknowledged. Alternatively, consider introducing a delay or a timelock when updating fixed-term parameters to avoid unexpected changes for users.
Comments / Resolution	

Issue_19	Interest on Moolah covers part of the interest owed to the broker
Severity	**Informational**
Description	It is stated that in markets where a broker is set, no interest will accrue in Moolah itself. However, there si nothing to prevent that, so it should be noted that if interest is unintentionally/intentionally enabled in both Moolah and the broker simultaneously, then during liquidations the interest accrued in Moolah will offset part of the interest owed to the broker. **Example:** Initial position: - Borrow assets in Moolah: 100e18 - Actual debt in broker (denormalized): 100e18 Pass some time and interest is accrued in both sides: - Borrow assets in Moolah: 105e18 - Actual debt in broker: 106e18 During full liquidation of that position, the protocol calculates the interest owed to the broker as: *interestToBroker = mulDivUp(repaidAssets, totalDebtBroker, totalDebtMoolah) - repaidAssets* *interestToBroker = mulDivUp(105e18, 106e18, 105e18) - 105e18 = 1e18* Although the broker should receive 6e18 in interest, only 1e18 is accounted for because the 5e18 interest accrued in Moolah effectively offsets it.
Recommendations	Consider acknowledging this behavior and ensure the IRM in Moolah is not activated for markets that use a broker.
Comments / Resolution	

Issue_20	refinanceMaturedFixedPositions doesn't validate the position
Severity	**Informational**
Description	As explained in another issue, liquidations can leave one or more positions below minLoan. Suppose any of these positions are refinanced into a dynamic loan that did not previously exist for the user. In that case, the fixed loan below minLoan (due to liquidation) is effectively transferred to the new dynamic loan. The consequences are transferred as well: the user will be forced to repay/borrow first his dynamic position to push it out of minLoan value before managing his fixed loans. This behavior is allowed because the function does not validate the resulting positions after the position's value transfer, which should not be permitted.
Recommendations	Consider adding a validation step to ensure that a new dynamic loan cannot be created below the minLoan threshold.
Comments / Resolution	

Issue_21	Repay function tries to transfer 0 tokens
Severity	**Informational**
Description	In the repay function for dynamic loans, the repayInterestAmt is first calculated and paid off. However, if multiple repays happen in the same block, this value can actually be 0 since there will be no new interest accrued. In such a situation, the contract will still try to transfer 0 tokens, which are not supported by some tokens and can revert.
Recommendations	Consider adding a repayInterestAmt>0 check before doing the interest transfer.
Comments / Resolution	

Issue_22	Missing events when adding/removing a fixed-term struct
Severity	**Informational**
Description	Within the updateFixedTermAndRate function, when updating a fixed term, an event FixedTermAndRateUpdated is emitted. However, when adding or removing fixed terms, no event is emitted.
Recommendations	Consider adding the events if you plan to keep track of these actions. Otherwise, this can be acknowledged.
Comments / Resolution	

Issue_23	BrokerMath::deductDynamicPositionDebt can be restricted to pure
Severity	**Informational**
Description	The function is set to public and can be changed to pure, as it neither modifies nor reads the contract state.
Recommendations	Consider changing it to pure.
Comments / Resolution	

RateCalculator

The RateCalculator contract implements the mechanism to accrue interest for dynamic loans, which has already been described in the Appendix in the LendingBroker section. In this contract, the Bot role can set an interest accrual rate at any time. Every second, the rate value increases by this amount. When loans are taken out, the borrow shares are calculated by dividing by this ever-increasing number. Thus, during repayment, a larger amount needs to be repaid to negate the borrow shares during issuance.

The contract implements an APY, where the rate is raised to the power of the time elapsed, to account for frequent accruals.

Privileged Functions

- batchSetRatePerSecond
- setRatePerSecond
- setMaxRatePerSecond
- registerBroker
- deregisterBroker

Core Invariants:

INV 1: Rate value must be monotonically increasing
INV 2: Rate must increase by the same percentage point in a duration, irrespective of how many times it is accrued in that duration.

Issue_24	Remove unused PAUSER_ROLE
Severity	**Informational**
Description	The RateCalculator contract implements a PAUSER role that is never used. *bytes32 public constant PAUSER = keccak256("PAUSER");*
Recommendations	Consider removing the unused PAUSER constant declared.
Comments / Resolution	

BrokerMath

The BrokerMath contract implements helper functions for the LendingBroker contract.

It implements an oracle function to scale the collateral price based on open interest used by the Moolah vault. Besides this, it implements functions to update the accounting during loan issuance, repayments, and liquidations, along with share-asset conversions.

Appendix: peek

The peek function implements the collateral oracle used by the Moolah contract. This function first checks the value of the collateral via an oracle and then scales down its price according to the accrued debt, following the expression below.

> collateralPrice = oraclePrice - AccruedDebt/TotalCollateral

Issue_25	Positions can be left stranded if minLoanValue is modified
Severity	High
Description	The checkPositionsMeetsMinLoan function iterates over each loan position and makes sure each of them is larger than the minimum Moolah loan amount. The issue is that if multiple positions are below this limit, the user is stranded. This is because no borrows or repays will be possible, since every function calls the _validatePositions function in the end. Thus, if there are 2 such borrow positions with their loans below the minimum limit, repaying one of them will fail due to the presence of the other, and the user will be completely stuck. This can happen if the Manager role of the Moolah vault changes the minLoanValue. If they raise this value such that multiple existing loans fall below this, the loan owners will be unable to repay their positions.
Recommendations	Consider checking the total loan value instead of each individual one against the minLoan. Otherwise, in the repay function, skip the _validatePositions check if a position is being paid off completely. This will allow users with multiple stuck positions to repay and free themselves.
Comments / Resolution	

Issue_26	Precision loss due to division before multiplication
Severity	Low

Description	In the getPenaltyForFixedPosition function, the contract first scales down the apr by RATE_SCALE and then multiplies by the time.
	
	`penalty = Math.mulDiv(`
	`    Math.mulDiv(repayAmt, _aprPerSecond(position.apr), RATE_SCALE, Math.Rounding.Ceil), // repayAmt * APR(per second)`
	`    timeLeft,`
	`    2,`
	`    Math.Rounding.Ceil`
	`  );`
	
	This leads to a precision loss since it's scaled down first and then multiplied.

Recommendations	Consider changing to the implementation present in the getAccruedInterestForFixedPosition function, where the scaling down is done after multiplying the apr per second by the time duration.
Comments / Resolution	

Issue_27	Unnecessary gas cost in checkPositionsMeetsMinLoan
Severity	Low
Description	On every operation occurring in the broker, the function _validatePositions is used, which in turn calls the function checkPositionsMeetsMinLoan on the BrokerMath library. However, checking the minLoan requirement for all positions is not required for every operation. For example, when a borrow is performed for a dynamic position and _validatePositions is called, there is no need to check the minLoan requirement for fixed positions. This creates unnecessary gas cost since now the call loops through up to 30 fixed positions. The same applies during fixed position creation, where all existing fixed positions and the dynamic position are checked for the minLoan requirement instead of only the fixed loan position being created.
Recommendations	Consider optimizing the minLoan requirement check to avoid unnecessary gas cost on every operation.
Comments / Resolution	

Issue_28	Optimize user fixed position retrieval from broker storage in refinanceMaturedFixedPositions
Severity	**Informational**

Description	Function refinanceMaturedFixedPositions retrieves the same user fixed positions array twice from the broker storage on Lines 280 and 301. This can be optimized by retrieving it only once at the start of the function.
	
	FixedLoanPosition[] memory _positions = broker.userFixedPositions(user);
	
	FixedLoanPosition[] memory fixedPositions = broker.userFixedPositions(user);

Recommendations	Consider retrieving the user's fixed position list only once to save gas.
Comments / Resolution	

BrokerInterestRelayer

The BrokerInterestRelayer contract supplies loan tokens to the Moolah vault, which are then given out as loans to LendingBroker users. This contract's main purpose is to use the protocol-generated interest revenue to provide liquidity to the vault.

Privileged Functions

- supplyToVault
- addBroker
- removeBroker

Core Invariants:

INV 1: Only authorized broker contracts can call this contract to supply liquidity
INV 2: Only add liquidity if the contract has more than minLoan tokens

Issue_29	MANAGER can drain or redirect existing interest and penalty payments to other markets
Severity	**Governance**
Description	Function supplyToVault retrieves the MARKET_ID from the msg.sender. This allows a manager to register a malicious broker to redirect the interest to another market with the same loan token or simply drain the interest/penalty payments sitting idle in the contract.
Recommendations	Consider acknowledging this issue.
Comments / Resolution	

Issue_30	supplyToVault can fail due to faulty minimum loan check
Severity	**High**

Description	The supplyToVault function checks if the balance in the contract is higher than the minLoan amount, and if so, transfers it to the Moolah vault.
	
	uint256 minLoan = MOOLAH.minLoan(MOOLAH.idToMarketParams(IBrokerBase(msg.sender).MARKET_ID()));
	if (balance >= minLoan) {
	
	The issue is that Moolah implements this check differently. In Moolah, first, the supply shares are calculated by rounding down the assets.
	
	if (assets > 0) shares = assets.toSharesDown(market[id].totalSupplyAssets, market[id].totalSupplyShares);
	
	Then, in _checkSupplyAssets, the supply assets are calculated by rounding down the shares.
	
	return uint256(position[id][account].supplyShares).toAssetsDown(
	market[id].totalSupplyAssets, market[id].totalSupplyShares
	) >= minLoan(marketParams);
	
	So there are two rounding downs involved, so the Moolah check is more stringent than the Relayer check, and requires a higher amount of assets to pass the minLoan check. Thus, the relayer check might pass, but the Moolah check might fail if the amount in question is just the right value. In this case, the whole transaction might fail. This can stop repayment and liquidation transactions, which is a major issue.

Recommendations	Consider wrapping the MOOLAH.supply call in a try-catch block to prevent it from reverting the transactions if not successful.

	Also, adjust the minLoan check so that a deposit is attempted only if the balance is some threshold amount above the minLoan amount. Another option would be to simulate the deposit transaction and then check if the assets controlled in Moolah are above the minLoan amount.
Comments / Resolution	

Issue_31	No way to specify marketId for collected revenue
Severity	**Medium**
Description	The same relayer contract can be used by multiple brokers to supply loan tokens to the Moolah vaults. The relayer picks the market to supply to by calling back the msg.sender. *MOOLAH.supply(MOOLAH.idToMarketParams(IBrokerBase(msg.sender).MARKET_ID()), balance, 0, vault, "");* The issue is that it sends any accumulated funds to the last calling market. If the contract does not contain enough funds, it will keep accumulating tokens in the contract until it reaches minLoan. So revenue generated from market A will be accumulated until a call from market B forces all that revenue into market B. Thus, the revenue generated by a broker/market will not flow to the same market.
Recommendations	Consider keeping track of the accumulated revenue on a per-broker basis.
Comments / Resolution	

Issue_32	Inefficient liquidity provision
Severity	Low
Description	The supplyToVault function does not supply to the vault if the contract's balance is below the minLoan set on the Moolah market. At first glance, this may seem fine, as the function would otherwise revert. However, minLoan is checked against the total supplied assets of the vault, not on the amount supplied at each tx. This creates a situation where, if the balance of the interest relayer is below minLoan but the total supplied assets for the vault exceed minLoan, the function will not supply to the vault. As a result, the vault cannot generate interest on that amount. Even though the loss is minimal, it is indeed an inefficient behavior.
Recommendations	It is recommended to check the assets already supplied by the vault before supplying. If the total supplied assets are below minLoan, then fallback to checking the contract's existing balance.
Comments / Resolution	

Moolah

The Moolah vault is the underlying lending protocol of the system. This vault has been covered in previous audits, and only the changes are in scope of this current audit.

The main change is the way the contract prices its tokens. Previously, the contract would directly call an oracle to get the prices, but in the current version, if a broker is defined for a market, it calls the peek function in the broker to get the price. Furthermore, if a broker contract is defined, only the broker is allowed to borrow from and repay to the Moolah vault.

Issue_33	Allowing direct calls to liquidate on moolah leads to incorrect accounting
Severity	**Low**
Description	Currently, the Moolah contract does not enforce that the caller of liquidate is the broker in the case where the market has an assigned broker. Since liquidations on LendingBroker must be initiated on the lendingBroker in order for proper accounting to take place, any direct call to liquidate on Moolah will result in incorrect accounting on the LendingBroker contract.
Recommendations	Consider enforcing that the caller is the broker if a broker is assigned for a given market.
Comments / Resolution	

PriceLib

The PriceLib contract fetched the price of the collateral and loan tokens. It fetched the price from an oracle. If a Broker is defined, then it fetches the price from the Broker contract instead.

No issues were identified in this contract.