BAIL security

Lista Dao
SlisBNB Provider

FINAL REPORT

December ‘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 - SlisBNB Provider - Audit Report
Website	lista.org
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/lista-dao/moolah/tree/5ad6716028a4ce43a38d46791ca9d1b2603100d2
Resolution 1	https://github.com/lista-dao/moolah/tree/ea9bb305395d475e5f9e1627eb07e1060dadc5de
Resolution 2	https://github.com/lista-dao/moolah/tree/a00d9e137d985044abeceab47ac4b4722027f018

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no changes made)	Failed resolution	Open
High	3	2		1		
Medium	4	1	1	2		
Low	8	2		6		
Informational	7	4		3		
Governance	1	1				
Total	23	10	1	12		

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

SlisBNBProvider

The SlisBNBProvider contract is a provider contract for the Moolah contract. The Moolah markets system allows the specification of certain providers for specific markets, who then become the sole collateral provider for that market.

This contract allows users to supply slisBNB token as collateral to select Moolah markets. The contract itself handles the underlying interactions with the Moolah contract and also keeps an account of the user's deposits. In exchange for the collateral, the user is minted immovable slisBNBx tokens to their account or that of a delegatee based on a fixed exchange rate. Users can then borrow against their positions by interacting directly with the Moolah contract. The accounting is also done during the liquidation flow to reflect the changes in the collateral amount. Yield generated from the underlying collateral is reflected via the convertSnBnbToBnb function call, which allows this contract to mint more slisBNBx tokens to the users/delegatees as the yield accrues.

The current audit focuses on a migration to a new separate minter contract. Previously, the tokens were accounted for and minted from this contract itself. In this code change, the new SlisBNBxMinter contract is introduced, which handles the accounting and minting. Special logic is added to zero out the accounted amounts in the SlisBNBProvider contract and fill in the correct amounts in the Minter contract, migrating the accounting state over there.

Appendix: Migration

In the _syncPosition function, if the slisBNBxMinter has already been set, the migration flow is entered if either userLp userReservedLp of the account is non-zero.

In this case, first, the accounted userTotalDeposit of the account is set to 0. Then _rebalanceUserLp is called, which basically treats this state as a complete withdrawal, burning all issued slisBNBx tokens and zeroing out all accounts. Then the syncDelegatee function is called in the minter contract to set the delegatee address. The userTotalDeposit value is then reset, and rebalance is called in the new function, which handles setting up the accounting as if it were a completely fresh account.

Privileged Functions

- setUserLpRate
- setMpcWalletCap
- removeMPCWallet
- addMPCWallet
- setSlisBNBxMinter

Core Invariants:

INV 1: After migration, userLp and userReservedLp should be 0

INV 2: The slisBNBx distribution should stay the same before and after migration, provided there was no change in the exchange rates

Issue_01	Liquidations may be DOSed
Severity	**High**
Description	Liquidations may be avoided because _syncPosition will revert if syncDelegatee attempts to sync a delegatee that is already set on the minter contract. A user can set delegatee on mthe inter by calling delegateAllTo and setting the same delegatee that is present on the provider before a liquidation. When syncDelegatee is called, a revert will occur in the following code snippet: *require(* *newDelegatee != address(0) && newDelegatee != delegation[account],* The same can also be triggered for any account by calling syncDelegation on the provider contract for that account. This will set a delegatee in the Minter contract, which will then lead to a revert during the accounting update.
Recommendations	Consider not reverting when setting delegatee to the same value, as this may cause issues with liquidations.
Comments / Resolution	Fixed by following recommendation.

Issue_02	getUserBalanceInBnb underflows if the exchange rate of convertSnBnbToBnb drops
Severity	**Medium**
Description	getUserBalanceInBnb may underflow if convertSnBnbToBnb returns a value that is smaller than the sum of userLp and userReservedLp. This will lead to liquidations and rebalances being DOSed.
Recommendations	Consider capping the value to 0 instead of underflowing
Comments / Resolution	Fixed by following recommendation.

Issue_03	Capped burns can interfere with multiple delegations
Severity	**Medium**
Description	When tokens are burnt, the contract uses the _safeBurnLp function, which burns only up to the holder's balance. This is done this way since a holder's balance can actually be lower than the accounted for balance via userLp in case of an exchange rate change. The issue is that in such a scenario, there will be incorrect accounting if there are multiple delegators for a delegatee. Say delegators A and B delegate 50 tokens each to C. So C should be minted 100 slisBNBx tokens. Now say due to a rate change, C only has access to 90 tokens actually. This would mean proportionally A and B control 45 tokens each. But if in this situation A does a withdrawal, it will burn 50 tokens from C. So C will actually only have 40 tokens remaining. So even though A should have proportionally only controlled 45 tokens, they were able to burn 50 tokens.
Recommendations	Consider either re-working the balance model into a share-ratio model, or make sure the userLp value stays synced with the contract balance with regular updates.
Comments / Resolution	Acknowledged.

Issue_04	Delegate mismatch between Provider and Minter contracts
Severity	**Medium**
Description	Both the Provider and Minter contracts allow setting delegations for the old and new flows. The issue is that users can thus have two different delegations in the two different contracts in the same system. Due to this, when we call rebalance on the minter in _syncPosition, the account we mint to will be a different address on the minter, and be out of sync with the set delegate on the provider. Since the delegate is only synced if either userLP and userReservedLp > 0, we use the value of the delegate on the minter instead of syncing the delegate from the provider. The Minter can mint funds to the delegatee even before migration, since any user can call syncUserModuleLp and mint pending yield to the Minter delegatee. Thus, accounts can have two different funded delegatees. Furthermore, the Provider delegatee can hijack control from the Minter delegatee by calling syncDelegation.
Recommendations	Contracts need to ensure that only a single address can be delegated to. Consider forbidding delegations if the position hasn't been migrated over yet.
Comments / Resolution	Partially fixed, the Provider::delegatee could not hijack anymore the Minter::delegatee, so Minter::delegatee is safe. But it still exists the scenario where two different delegatees are set for the same non-migrated account, one in the minter and the other in the provider.

Issue_05	Incorrect conditional leads to stale delegation
Severity	**Medium**
Description	The following conditions must be met in order for syncing of delegation from the provider and the minter to proceed: *if (delegatee != address(0) && targetDelegatee != delegatee && delegatee != account)* This check is erroneous, this occurs because the condition will not pass if delagatee == account. However, it can occur that delagatee on the provider is account, but on the minter, it is some other address or even address(0). Therefore, the condition incorrectly assumes that since delagatee on the provider is account, then it must be so on the minter.
Recommendations	Consider removing the check delagatee != account as it restricts syncing of delegation.
Comments / Resolution	Acknowledged.

Issue_06	supplyCollateral will revert if users input any data
Severity	**Low**
Description	supplyCollateral allows users to specify data, but does not include the callback function in the contract, therefore leading to a revert each time.
Recommendations	Consider acknowledging this issue.
Comments / Resolution	Acknowledged.

Issue_07	Removing an MPC wallet changes the order of the array
Severity	**Low**
Description	When an MPC wallet is removed from the mpcWallets array, the last wallet in the array is moved into the index of the removed wallet. This reordering can be unfair, as a wallet that was originally positioned to be filled last may now be filled earlier than intended. *Note: wallets with a balance > 0 cannot be removed.* Example scenario: Four wallets are inserted with the following intended order: mpcWallets[0].balance = 10e18 → currently being filled mpcWallets[1].balance = 0 mpcWallets[2].balance = 0 mpcWallets[3].balance = 0 If the second wallet (mpcWallets[1]) is removed, mpcWallets[3] will be moved to index 1. This causes the wallet that was originally meant to be filled last to be filled immediately after the first wallet's cap is reached.
Recommendations	If the ordering of wallets is important to protocol logic, consider adding a mechanism to preserve the original order when removing a wallet from the middle of the array. However, implementing this safely can be complex and may introduce new risks. Acknowledging this behavior may therefore also be acceptable.
Comments / Resolution	Acknowledged.

Issue_08	Frequent outdated participation values
Severity	**Low**
Description	The current design mints and burns LP_tokens / SlisBNBx (BNB-equivalent tokens) to track each user's participation in the Binance Launchpool. However, the value of these tokens changes frequently due to: - Exchange rate updates driven by accrued rewards in StakerManager - Changes in fee / userLpRate, and - Discount changes. Because user balances are only updated when their position is explicitly synced, users can frequently hold outdated participation values. Depending on timing, this can either advantage or disadvantage them, leading to an inconsistent representation in the Launchpool. While a permissionless function exists to rebalance any account, the protocol cannot realistically rely on calling it for all users after every exchange rate or parameter change, especially when the user base is large. It is unclear how strictly the Launchpool requires up-to-date participation values at all times, but if accuracy is important to the system's integrity, the current design may introduce persistent inconsistencies.
Recommendations	Ensure that balances are kept as up-to-date as possible
Comments / Resolution	Acknowledged.

Issue_09	UserLpRate changes don't affect immediately
Severity	**Low**
Description	When userLpRate is updated, all position balances should increase/decrease accordingly when syncing the position. However, a user can simply choose not to sync their position when userLpRate increases (which would reduce their balance due to higher fees). This allows the user to temporarily retain an inflated share of the launchpool. A permissionless function exists that allows anyone to sync any user's position, and the protocol can proactively use this to update all users. However, this approach may be impractical when the number of users is large. This behavior can also be exploited to maximize a user's participation amount by front-running the userLpRate update, syncing their position before the new (higher) fee takes effect while still benefiting from the updated exchange rate (if it has increased since their last sync).
Recommendations	Consider acknowledging this issue.
Comments / Resolution	Acknowledged.

Issue_10	removeMPCWallet may be DOSed
Severity	**Informational**
Description	removeMPCWallet may not be possible since setting a wallet cap to 0 is not allowed, and if the MPC wallet does have 0 balance, a rebalance by a frontrunner may fill the wallet with balance again.
Recommendations	Consider allowing a wallet cap to be set to 0; otherwise, removal of wallets may only be possible when withdrawals happe,n and even the,n they can be dosed.
Comments / Resolution	Fixed by following recommendation.

SlisBNBxMinter

The SlisBNBxMinter contract now does the accounting and slisBNBx token minting, which was previously done by the slisBNBProvider contract. This contract further allows the integration of multiple configurable modules, thus allowing multiple such providers to hook up the same minter with different individual configurations.

Privileged Functions

- setMpcWalletCap
- removeMPCWallet
- addMPCWallet
- updateModules

Issue_11	Collateral is locked if the provider is disabled from minter but not removed from the market
Severity	**Governance**
Description	If a module is disabled in the SlisBNBxMinter contract, all actions involving that module/provider become blocked; no minting or burning can occur. If the provider is not removed from the Moolah market before being disabled, users will no longer be able to withdraw their collateral directly through Moolah. The only remaining option would be to withdraw through the provider itself, but this will revert because the provider is disabled and can no longer interact with the minter.
Recommendations	Ensure that any module being disabled is also removed from the Moolah market to prevent collateral from becoming locked. Or allow that provider to interact with another minter contract.
Comments / Resolution	Fixed by changing a disabled market simply to a market with 100% discount. This allows the market to stop minting new tokens and also allows users' funds to be withdrawn at any time.

Issue_12	userTotalBalance can underflow
Severity	High

Description	Due to changes in exchange rates, it can happen that the actual slisBNBx balance in the contract is lower than the recorded balance. Due to this, the _safeBurnLp function does a conservative burn and burns only up the available amount, preventing a revert in case of insufficient funds.
	
	The issue is that this contract also keeps track of userTotalBalance, which does a direct subtraction.
	
	_safeBurnLp(holder, oldUserLp - newUserLp);
	userTotalBalance[account] -= oldUserLp - newUserLp;
	
	So in such a situation, where the holder has fewer tokens than oldUserLp-newUserLp, the burn call will process fine, but the next subtraction will underflow. This will lead to a revert.
	
	A similar revert can also be encountered in the _burnFromMPCs function, which does a direct uncapped burn. If the MPC holds an insufficient number of tokens, this burn can revert.

Recommendations	If a conservative burn is required, then the same logic must also be applied to the userTotalBalance update. The subtractions should saturate to 0, and a similar update to userTotalBalance should also be carried out in the _delegateAllTo function, which also does a saturated burn.
Comments / Resolution	Partially fixed, _burnFromMPCs still will revert if not enough tokens are present, however userTotalBalance now prevents the underflow revert.
Comments / Resolution 2	Fixed, _burnFromMPCs now uses _safeBurnLp which will prevent reverts in case of underflow. However an edge case exists, let us assume the following scenario. userTotalBalance = 100 oldUserLp = 100

	newUserLp = 10 cut = 100-10 = 90 _safeBurnLp is called to burn 90 slisBNBx, but it only burns 80 due to the delegatee not holding enough tokens. However, the returned value is ignored. As a result, the userTotalBalance is decreased to 10 even though the delegatee does not hold any tokens. A similar scenario is possible in _burnFromMPCs and setMpcWalletCap, where the _safeBurnLp return value is ignored as well. If this is unacceptable it can be fixed, otherwise it can be acknowledged as slisBNBx tokens are non-transferable thus the risk is minimal.

Issue_13	userTotalBalance can be deducted multiple times, leading to lower values
Severity	**High**
Description	In the _delegateAllTo function, the holder's LP tokens are burnt. If insufficient tokens are burnt, userTotalBalance is adjusted to reflect the current total balance. *if (actualBurned != userTotalBalance[account]) {* *// adjust userTotalBalance if actual burned is less than expected* *userTotalBalance[account] = actualBurned;* *}* The issue is that userTotalBalance is reduced without changing the stored oldUserLp value. Due to this, there will be another reduction when _rebalanceUserLp is called. Say the slisBNBx amount was originally 100 and then reduced to 90. Then _delegateAllTo will burn 90 tokens and set userTotalBalance to 90 instead of 100. But oldUserLp is still set to the old value of 100. So when _rebalanceUserLp is called, newUserLp is 90 while oldUserLp is 100, so userTotalBalance will further get reduced by 10, to 80. So userTotalBalance gets reduced to 80 due to a drop of just 10.
Recommendations	Consider syncing the LP position before burning the LP in _delegateAllTo.
Comments / Resolution	Acknowledged, "The BSC mainnet migration has been completed. We ensured that userTotalBalance equals the previous oldUserLp after migration, and both oldUserLp and oldReservedLp have been reset to 0."

Issue_14	Fee rate increase causes partial liquidations to revert when MPC caps are full
Severity	**Low**
Description	The SlisBNBxMinter and SlisBNBProvider enforce a hard cap on MPC wallets within the _mintToMPCs function. When newReservedLp (fees) are generated, the system attempts to mint slisBNBx to the registered MPC wallets. If all wallets reach their defined cap, the function reverts. If the feeRate increases or the discount decreases , a user's newReservedLp may increase even during a partial liquidation (where the principal stakedAmount decreases). If the MPC wallets are near capacity, the liquidation transaction calls _mintToMPCs and reverts. This prevents bad debt from being cleared.
Recommendations	Consider allowing liquidations to go through even when all wallets are full
Comments / Resolution	Partially Resolved. Liquidations are now possible even if the mint cap is reached. However, the slisBNBx tokens are no longer burnt in case the mint cap is reached.
Comment / Resolution 2	Acknowledged, the try-catch logic has been removed and reverted to the original implementation. The previous fix introduced unnecessary complexity, the protocol will monitor caps regularly to avoid the issue.

Issue_15	DoS on bulkSyncUserModules via frontrunnning
Severity	**Low**
Description	The bulkSyncUserModules function synchronizes multiple user accounts in a single transaction. For each account, it internally calls syncUserModuleLp. However, syncUserModuleLp will revert if the position for that account has already been synchronized. This allows an attacker (or any user) to frontrun a bulkSyncUserModules call by synchronizing just one of the accounts in the array. When the original transaction executes, it will revert due to that single already-synced account, effectively griefing the caller and causing a DoS.
Recommendations	Allow bulkSyncUserModules to run even if the accounts are already synced
Comments / Resolution	Fixed by following recommendation.

Issue_16	No way to remove a compromised MPC wallet
Severity	**Low**
Description	If an MPC wallet becomes compromised, there is no way to remove it when its balance is > 0, and it is positioned in the middle of the mpcWallets array. To remove it, all wallets before it in the array would first need to be fully emptied, which may be impractical. There is a partial mitigation: its cap can be reduced to the minimum value (1 wei, since the cap cannot be set to zero). This effectively minimizes the wallet's participation by reallocating its minting share to other wallets. But still, the compromised wallet remains in the array with a non-zero cap and cannot be fully removed.
Recommendations	This can be fixed by allowing to setting a cap of zero. In that way, the wallet could be removed if it is in the middle of the array and their wallets above have their balances > 0.
Comments / Resolution	Partial resolution, the issue still persists in the SlisBNBProvider contract as caps cannot be set to 0.
Comments / Resolution 2	Fixed, caps can now be set to 0 on the SlisBNBProvider contract.

Issue_17	Attacker can consume MPC wallet caps to DOS deposits
Severity	**Low**
Description	An attacker can consume MPC wallets cap by sandwiching user deposits with collateral deposit and withdraw calls. Since the _mintToMPCs function implements the check below, the call reverts as leftToMint will be non-zero due to not being consumed by any MPC wallet that has hit its cap. *require(leftToMint == 0, "Not enough MPC wallets to mint");*
Recommendations	Consider acknowledging this risk and monitor such griefing attacks to take preventative measures such as increasing wallet caps.
Comments / Resolution	Acknowledged.

Issue_18	Unbounded mpcWallets array leads to DoS
Severity	**Informational**
Description	The SlisBNBxMinter iterates over the entire mpcWallets array in _mintToMPCs and _burnFromMPCs during every user rebalance, deposit, and withdrawal. There is no hard limit on the number of wallets that can be added via addMPCWallet. If the mpcWallets array grows large enough to consume the block gas limit during these loops, all core protocol interactions will revert. The same issue also exists in the SlisBNBProvider.
Recommendations	Introduce a constant maximum limit in addMPCWallet to prevent the array from growing beyond a safe size.
Comments / Resolution	Acknowledged.

Issue_19	SlisBNBxMinter.delegateAllTo transfers user balance at stale rates
Severity	Informational
Description	The SlisBNBxMinter.delegateAllTo function transfers the user's currently recorded userTotalBalance to a new delegatee without triggering a rebalance before this transfer. This means the delegation occurs using potentially outdated exchange rates and fee configurations. The new delegatee receives a "stale" balance that does not include pending rewards or fee adjustments that have accrued since the last interaction. In contrast, the SlisBNBProvider implementation always rebalanced the user before delegating, ensuring the new delegatee received the precise, up-to-date value of the position.
Recommendations	Consider rebalancing during SlisBNBxMinter.delegateAllTo to ensure the new delegatee receives the current, accurate balance.
Comments / Resolution	Acknowledged.

Issue_20	The fee is rounded down
Severity	Informational
Description	When calculating the fee from the discounted amount, it is rounded down. Therefore, in some cases, the user will mint SLisBNBx tokens without the fee being applied.
Recommendations	Change the fee rounding to ensure a fee is always charged for MPC wallets.
Comments / Resolution	Fixed by following recommendation.

Issue_21	Incorrect inequality in _mintToMPCs
Severity	Informational
Description	_mintToMPCs will enter the if branch even if the balance cap is reached because of incorrect inequality, *if (balance <= wallet.cap) {* *uint256 toMint = balance + leftToMint > wallet.cap ? wallet.cap - balance : leftToMint;* The inequality should be < instead of <= since when the balance is equal to the wallet.cap, there is no capacity left, and thus entering the if branch is redundant.
Recommendations	Consider changing the inequality in _mintToMPCs from <= to <.
Comments / Resolution	Fixed by following recommendation.

Issue_22	Tokens are not burned if provider is only removed from Moolah
Severity	**Informational**
Description	If a provider is removed from a market in Moolah but not deactivated in the Minter (by setting a 100% discount), users may still retain SlisBNBx tokens if they synchronise only with the minter and not with the provider. In this case, the minter relies on the provider's last recorded state (userTotalDeposit). If the exchange rate increases after the provider is removed, syncing through the minter may result in additional tokens being minted rather than burned, leaving outdated participation tokens in circulation.
Recommendations	When deactivating a provider market, make sure to deactivate it in Minter as well
Comments / Resolution	Acknowledged.

Issue_23	Free module can not be added at the beginning
Severity	**Informational**
Description	When updating a module, the following check is always present: *require(_config.discount != config.discount || _config.feeRate != config.feeRate, "no changes detected");* As a result, a new module configured with both discount = 0 and feeRate = 0 cannot be added directly. To introduce a free module, it must first be configured with a non-zero discount or fee, and then updated again to set both values to zero.
Recommendations	If the protocol does not intend to introduce free modules, this behaviour can be acknowledged. Otherwise, consider allowing an initial configuration of zero values.
Comments / Resolution	Fixed by adding a new addModule function which allows free modules.