BAIL security

Lista Dao
Smart 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 – SmartProvider - 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/a00d9e137d985044abeceab47ac4b4722027f018

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no changes made)	Failed resolution	Open
High						
Medium	3		1	2		
Low	2	1		1		
Informational	4	1		3		
Governance						
Total	9	2	1	6		

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

SlisBNBxMinter

The SlisBNBxMinter contract is a new contract designed to standardize the minting process for slisBNBx across multiple contracts within the ListaDAO ecosystem. This replaces the earlier approach, in which slisBNBx was minted separately by several different contracts.

The slisBNBx token is a non-transferable receipt token that tracks a user's deposited collateral, with each token corresponding to one BNB-worth of collateral. Holding slisBNBx allows users to participate in supported programs, such as Binance Launchpool, while their underlying collateral remains locked in the protocol.

Any contract in the ListaDAO ecosystem that intends to mint or burn slisBNBx through the slisBNBxMinter must first be manually registered as a module by the MANAGER.

Each registered module is required to implement a getUserBalanceInBnb function, which the minter uses to determine a user's balance (expressed as an equivalent BNB amount) within that module. Modules may invoke rebalance or syncDelegatee on the minter to sync the user's state, although these calls are not required for all integrations. Calling syncUserModuleLp or bulkSyncUserModules on the minter directly is permissionless and will update a user's slisBNBx balance based on their current balance in the modules passed to the function.

The slisBNBx minted from each module has two components: the feePart and the userPart. The percentage allocated to these two components is configured in each module.

The feePart is minted to MPC wallets configured in the minter. When the feePart for a user increases, slisBNBx is minted to these wallets up to a maximum amount configured for each wallet. When it decreases, slisBNBx is burned from these wallets.

The userPart is minted to the user, or to the user's delegate if they set one. Delegation applies to the user's entire balance, so the delegate receives all of the user's slisBNBx. When a user changes their delegate, the contract burns the full amount from the old delegate and mints the same amount to the new one.

Privileged Functions

- setMpcWalletCap
- removeMPCWallet
- addMPCWallet
- updateModules
- _authorizeUpgrade

Core Invariants:

INV 1: A module's discount and fee rates cannot exceed 1e6
INV 2: Each MPC wallet.balance never exceeds its wallet.cap
INV 3: The rebalance and syncDelegatee functions can only be called by enabled modules
INV 4: Lowering an MPC wallet's cap will redistribute the necessary slisBNBx to other wallets

Issue_01	_safeBurnLp is not used consistently
Severity	**Medium**
Description	The _safeBurnLp helper function burns a user's slisBNBx, but it first checks the user's balance to ensure it doesn't burn more than they actually hold. So, if the internal accounting ever becomes out of sync with the user's real slisBNBx balance, _safeBurnLp caps the burn at the user's balance instead of reverting. However, this protection is not applied everywhere. If the internal accounting can indeed be larger than the user's real balance, the following locations could run into issues: 1. In _rebalanceUserLp, the userTotalBalance mapping value is always reduced by the full burn amount, even if _safeBurnLp ends up burning less. If the accounting is out of sync, this decrement could underflow. 2. In _burnFromMPCs and setMpcWalletCap, the contract burns slisBNBx directly from wallet.walletAddress without using _safeBurnLp. If the address holds fewer tokens than the internal accounting assumes, the burn would revert. 3. In _delegateAllTo, the minted amount could be less than userTotalBalance since only actualBurned amount is minted. But future delegation changes will incorrectly reduce userTotalBalance itself from the delegatee In practice, the accounting might never get out of sync with the actual slisBNBx balance. The SlisBNBxMinter contract itself doesn't appear able to mint or burn in a way that would cause a mismatch. But other contracts can also mint and burn slisBNBx, and if any of them burn more from a user than they mint, the locations above could revert and cause issues. Since the behavior of the other slisBNBx minters isn't guaranteed to be consistent, it would be safer to handle this scenario directly in the SlisBNBxMinter code.
Recommendations	Ensure the code safely handles any scenario where the accounting may go out of sync with the user's slisBNBx balance. The _rebalanceUserLp case can be fixed by adding underflow protection

	for the decrement, the _burnFromMPCs/setMpcWalletCap case can be fixed by calling _safeBurnLp instead of burning directly, and _delegateAllTo can be fixed by minting the entire amount instead of actualBurned
Comments / Resolution	Partially resolved 1. Underflow is now prevented 2. Burning is now always done via _safeBurnLp. But in case the amount actually burned is lower, leftToBurn is still reduced by the full amount (this is related with the comment about 3. below) 3. The issue of difference b/w minting and burning is handled. But if the actual SlisBNBx balance can be reduced below sum of userTotalBalances delegated to that address, then the behaviour is still not ideal since unless _rebalanceUserLp is called, the last users will have their userTotalBalance reduced while the early users will keep their full balance without suffering any deduction

Issue_02	MPC wallet cap limits can trigger reverts
Severity	**Medium**
Description	The _mintToMPCs function will revert if all MPC wallets have reached their wallet.cap. This can cause rebalancing to revert, which would be especially problematic during time-sensitive liquidations. Even though liquidations typically result in burning slisBNBx, changes to the discount and fee rates of a module can technically increase the portion that needs to be minted to MPCs, even when the overall slisBNBx supply stays the same or decreases.
Recommendations	Ensure that MPC wallet caps are set high enough that such reverts don't happen.
Comments / Resolution	Acknowledged.

Issue_03	SlisBNBx balances can become stale
Severity	**Low**
Description	In most module implementations, the BNB-denominated value of a user's position will change even when the position itself is not touched. This applies to any module that holds something other than pure BNB. For example, if a module holds LP tokens, the BNB value of those LP tokens can change even when the user hasn't interacted with their position. However, the SlisBNBxMinter will typically only sync a user's slisBNBx balance when the position is modified, such as during deposits or withdrawals. It does not automatically track changes in the position's value unless these actions occur. This means the user's SlisBNBx balance can diverge over time from the actual BNB value of their positions. Anyone can call syncUserModuleLp or bulkSyncUserModules to resync other users' positions, but there is no guarantee this will occur.
Recommendations	Ensure that there is a system that resyncs a user's slisBNBx balance if it diverges from the actual BNB value of their positions.
Comments / Resolution	Acknowledged.

Issue_04	bulkSyncUserModules batch call can revert due to a single inner revert
Severity	**Informational**
Description	The bulkSyncUserModules batch function calls syncUserModuleLp on each of the entries individually. Since syncUserModuleLp can revert, a single entry can cause the entire batch call to revert
Recommendations	Refactor syncUserModuleLp to not revert
Comments / Resolution	Fixed. Now the bulkSyncUserModules function invokes private variant of syncUserModuleLp that returns instead of reverting.

Issue_05	setMpcWalletCap and removeMPCWallet operating on indices can be error-prone
Severity	**Informational**
Description	The setMpcWalletCap and removeMPCWallet functions operate based on the provided index of an MPC wallet in the mpcWallets array. This can be error-prone if multiple actions are batched together or if two different MANAGER transactions occur around the same time, and the intended targets shift their indices in the array.
Recommendations	Given that this behavior is inherited from the existing SlisBNBProvider code, consider acknowledging this issue without making code changes, and keep this behavior in mind when using these functions.
Comments / Resolution	Acknowledged.

Issue_06	No limit on mpcWallets array size
Severity	**Informational**
Description	The mpcWallets array can have a large number of elements added to it through addMPCWallet calls from the MANAGER role. If the array grows large enough, it may prevent actions like _burnFromMPCs from completing due to running out of gas when iterating over the entire array. However note that addMPCWallet itself iterates over the full array before adding new elements, which could make this less likely to be an issue.
Recommendations	Keep this behavior in mind and consider enforcing an explicit upper bound on the array length in the code.
Comments / Resolution	Acknowledged.

SmartProvider

The SmartProvider contract is a provider for the Moolah lending codebase. It enables users to deposit LP tokens from a StableSwapPool in order to mint StableSwapLPCollateral to be used as collateral for borrowing. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope.

The in-scope changes were the modifications required for the SmartProvider to integrate with the SlisBNBxMinter contract.

This includes introducing internal tracking of the user's total collateral across all markets and a new _syncPosition function, which updates the internal balances and calls into the minter to sync the user's slisBNBx balance. This function is called at the end of all operations that may have changed the user's collateral balance.

Another addition is the getUserBalanceInBnb function, which computes the BNB value of the user's total LP collateral. This value is used by the SlisBNBxMinter.

Privileged Functions

- setSlisBNBxMinter

Core Invariants:

INV 1: userTotalDeposit[account] is equal to the sum of userMarketDeposit[account][id] across all ids
INV 2: _syncPosition is called at the end of any action that changes the user's collateral balance

Issue_07	Manipulated pool state will inflate getUserBalanceInBnb
Severity	**Medium**
Description	The getUserBalanceInBnb function calculates the value of a user's LP tokens in BNB terms. It does this by calling calc_coins_amount to determine the underlying token amounts redeemable at the current pool state, converting each amount to USD with an oracle, summing the two USD values, and then dividing that total by the BNB USD price. This approach relies on the pool's current state, which makes it vulnerable to price manipulation. If the pool is temporarily pushed into an imbalanced state that doesn't reflect the fair value of the two tokens, the function can return an artificially high result. This is a known issue with using underlying pool reserves when pricing LP tokens. There are some safeguards in place for this issue. The StableSwapPool includes checkPriceDiff calls that revert if the pool price deviates too far from a configured threshold. And if manipulation succeeds, any inflated value can be corrected once the pool returns to a normal state and the user has their SlisBNBx balance resynced. However, since inflated results could temporarily mint too much SlisBNBx to a user, it would be safer to use a manipulation-resistant pricing method.
Recommendations	Consider changing the pricing formula to be manipulation-resistant. One such option is already implemented in the peek function, which uses the pool's virtual price to determine the LP token's USD value.
Comments / Resolution	Acknowledged.

Issue_08	Reentrancy concerns in getUserBalanceInBnb
Severity	**Low**
Description	The getUserBalanceInBnb function is a view function that returns the BNB value of the user's total LP collateral. It's utilized by the SlisBNBxMinter, and it uses the current StableSwapPool state to value the LP tokens. The function does not trigger any reentrancy guards in either the SmartProvider or the underlying StableSwapPool. If the StableSwapPool is mid-execution when getUserBalanceInBnb is called, it can return incorrect values. For example, if a user gains control flow during a remove_liquidity call, the pool balances are only partially updated, which would distort the result. The feasibility of this issue is low, because native token transfers forward only 23_000 gas, so abusing this would likely require a token with callbacks (e.g. ERC777). However, it would be safer for getUserBalanceInBnb to revert if the underlying pool is in a reentrant state, since this is never intended. If the SmartProvider is mid-execution during getUserBalanceInBnb, the situation is more nuanced. Some reentrant calls are intentional, for example when the SmartProvider calls rebalance and the minter calls back into getUserBalanceInBnb. Other cases could be unintended, such as if a user gains control flow during a transferOutTo call and then invokes the minter directly while the SmartProvider is in a reentrant state. In practice, this does not appear to be an issue: the SmartProvider re-syncs its state at the end of each relevant function, and the only state that matters (userTotalDeposit) is never inflated even in temporary states. At worst, getUserBalanceInBnb would read a value that was already valid moments earlier, or one that will become the correct value once the function completes. So, not checking the SmartProvider's reentrancy guard does not appear to be a major concern, which is helpful because reentrancy is legitimately needed in some cases.
Recommendations	Consider having getUserBalanceInBnb revert if the StableSwapPool

	is in a reentrant state. This could be accomplished by using the pool's get_virtual_price function. Also, keep the reentrancy behavior relative to the SmartProvider's reentrancy guard in mind.
Comments / Resolution	Fixed. Now reverts on re-entrancy since get_virtual_price is invoked.

Issue_09	Unused event
Severity	**Informational**
Description	The SmartLiquidation event is currently unused.
Recommendations	Either remove the code or emit it inside liquidation.
Comments / Resolution	Fixed. The event is removed.