BLOCKSEC

Security Audit Report for Lista Lending

Date: December 1, 2025 **Version:** 1.0
Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Lista
Target	Lista Lending

Version History

Version	Date	Description
1.0	December 1, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-12-01 11:27:40+0000

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of Lista Lending of Lista.

Lista Lending adds SlisBNBProvder, ETHProvider, BNBProvider, SmartProvider on top of Moolah and MoolahVault. SlisBNBProvider allows users to mint clisBNB while they deposit slisBNB collateral. SmartProvider allows users to participate in Moolah and MoolahVault using dex LP. SlisBNBxMinter allows users to stake supported collateral tokens (e.g. slisBNB and smart-LP) and receive slisBNBx in return to participate in Binance Launchpool.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- src/provider/SlisBNBProvider.sol
- src/provider/SmartProvider.sol
- src/utils/SlisBNBxMinter.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
moolah	Version 0	179f0830a8fa47bd0d985506b6fd3004c4cf2395
	Version 1	5ad6716028a4ce43a38d46791ca9d1b2603100d2
	Version 2	31d2065811dc45dfa5bd4856c675a99476351 48d

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/lista-dao/moolah

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

- Access control
- Permission management
- Whitelist and blacklist mechanisms
- Initialization consistency
- Improper use of the proxy system
- Reentrancy
- Denial of Service (DoS)
- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism

∗ Economic and incentive impact

1.3.2 Additional Recommendation

∗ Gas optimization
∗ Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ **Likelihood**	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **three** recommendations and **two** notes.

- Medium Risk: 3
- Recommendation: 3
- Note: 2

ID	Severity	Description	Category	Status
1	Medium	Immutable delegatee assignment once `slisBNBxMinter` is set	Security Issue	Confirmed
2	Medium	Conflicting delegatee synchronization between contracts	Security Issue	Confirmed
3	Medium	Module disablement halts liquidation and collateral withdrawal	Security Issue	Fixed
4	-	Redundant code	Recommendation	Fixed
5	-	Potential zero token minting in function `_mintToMPCs()`	Recommendation	Confirmed
6	-	Add zero value check in wallet configuration	Recommendation	Fixed
7	-	LP token value calculation depends on the function `get_virtual_price()`	Note	-
8	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Immutable delegatee assignment once `slisBNBxMinter` is set

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description The contract `SlisBNBProvider` implements the function `delegateAllTo()`, which allows users to modify their delegation preferences. However, the function reverts when `slisBNBxMinter` is set. This design contradiction prevents users from updating their delegation targets even when legitimate operational requirements change, such as when a `delegatee` becomes inactive or suboptimal.

```
function delegateAllTo(address newDelegatee) external {
  require(slisBNBxMinter == address(0), "not supported");
  require(
    newDelegatee != address(0) && newDelegatee != delegation[msg.sender],
    "newDelegatee cannot be zero address or same as current delegatee"
  );
  // current delegatee
  address oldDelegatee = delegation[msg.sender];
```

```
    // burn all lpToken from account or delegatee
    _safeBurnLp(oldDelegatee, userLp[msg.sender]);
    // update delegatee record
    delegation[msg.sender] = newDelegatee;
    // clear user's lpToken record
    userLp[msg.sender] = 0;
    // rebalance user's lpToken
    _rebalanceUserLp(msg.sender);

    emit ChangeDelegateTo(msg.sender, oldDelegatee, newDelegatee);
  }
```

Listing 2.1: src/provider/SlisBNBProvider.sol

Impact This design contradiction prevents users from updating their delegation targets even when legitimate operational requirements change, such as when a `delegatee` becomes inactive or suboptimal.

Suggestion Revise the logic accordingly.

Feedback from the project The project team stated that once the `SlisBNBxMinter` contract is fully configured, it becomes the sole entry point for managing user delegation targets. Therefore, the fact that the `SlisBNBProvider` contract cannot update user delegation targets is by design.

2.1.2 Conflicting delegatee synchronization between contracts

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description If a user specifies a `newDelegatee` through the function `delegateAllTo()` in the contract `SlisBNBxMinter`, when anyone invokes the function `syncDelegation()` in the contract `SlisBNBProvider`, it will trigger the function `syncDelegatee()` in the contract `SlisBNBxMinter`. This will revert the user's newly set `newDelegatee` back to the old delegatee recorded in `SlisBNB-Provider`. Furthermore, according to Issue-1, the old delegatee recorded in `SlisBNBProvider` is either the user themselves or an outdated delegatee, and it cannot be modified.

```
function syncDelegatee(address account, address newDelegatee) external {
  require(moduleConfig[msg.sender].enabled, "unauthorized msg.sender");

  _delegateAllTo(account, newDelegatee);
}

/**
 * @dev delegate all slisBNBx to given address without rebalancing
 * @param newDelegatee new target address of collateral tokens
 */
function delegateAllTo(address newDelegatee) external {
  _delegateAllTo(msg.sender, newDelegatee);
}

```

```
function _delegateAllTo(address account, address newDelegatee) internal {
  require(
    newDelegatee != address(0) && newDelegatee != delegation[account],
    "newDelegatee cannot be zero address or same as current delegatee"
  );
  // current delegatee
  address oldDelegatee = delegation[account];
  if (oldDelegatee == address(0)) {
    oldDelegatee = account;
  }
  // burn all slisBNBx from account or delegatee
  uint256 actualBurned = _safeBurnLp(oldDelegatee, userTotalBalance[account]);
  // update delegatee record
  delegation[account] = newDelegatee;
  // mint all burned slisBNBx to new delegatee
  SLISBNB_X.mint(newDelegatee, actualBurned);

  emit ChangeDelegateTo(account, oldDelegatee, newDelegatee, actualBurned);
}
```

Listing 2.2: src/utils/SlisBNBxMinter.sol

```
function syncDelegation(address[] calldata accounts) external {
  for (uint256 i = 0; i < accounts.length; i++) {
    address account = accounts[i];
    address delegatee = delegation[account];
    if (delegatee != address(0)) {
      // write data to slisBNBxMinter
      ISlisBNBxMinter(slisBNBxMinter).syncDelegatee(account, delegatee);
    }
  }
}
```

Listing 2.3: src/provider/SlisBNBProvider.sol

Impact This conflict creates unreliable delegation management where user preferences can be arbitrarily overridden by outdated records.

Suggestion Revise the logic accordingly.

Feedback from the project The project team stated that the existing delegation records are few, so this issue will not have a significant impact. At the same time, the project team will ensure that all existing delegation data is synced to the `SlisBNBxMinter` contract before opening the delegation portal on the frontend.

2.1.3 Module disablement halts liquidation and collateral withdrawal

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `SlisBNBxMinter` implements the function `updateModules()`, which allows the `MANAGER` role to enable or disable individual modules within the system. If a `MANAGER` disables a module by setting its `enabled` parameter to false, this action would halt all functionality associated with that module's `rebalance` and `syncDelegatee` operations.

This configuration change would directly prevent the liquidation processes and collateral withdrawals within dependent contracts such as `SlisBNBProvider` and `SmartProvider`, which rely on these module functions.

```
function updateModules(address[] calldata _modules, ModuleConfig[] calldata _configs) external
    onlyRole(MANAGER) {
  require(_modules.length == _configs.length, "modules and configs length mismatch");

  for (uint256 i = 0; i < _modules.length; i++) {
    address module = _modules[i];
    ModuleConfig memory config = _configs[i];
    require(module != address(0), "module is the zero address");
    require(config.feeRate <= DENOMINATOR, "userLpRate invalid");
    require(config.discount <= DENOMINATOR, "discount invalid");
    ModuleConfig memory _config = moduleConfig[module];
    require(
      _config.discount != config.discount || _config.feeRate != config.feeRate || _config.enabled
          != config.enabled,
      "no changes detected"
    );

    moduleConfig[module] = config;
    emit ModuleConfigUpdated(module, config.discount, config.feeRate, config.enabled);
  }
}
```

Listing 2.4: src/utils/SlisBNBxMinter.sol

Impact This configuration change would directly prevent the liquidation processes and collateral withdrawals within dependent contracts such as `SlisBNBProvider` and `SmartProvider`, which rely on these module functions for proper collateral management.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There is an unused event. It is recommended to remove it for better code readability.

```
event SmartLiquidation(
  address indexed liquidator,
  address indexed collateralToken,
  address dexLP,
  uint256 seizedAssets,
  uint256 minAmount0,
    uint256 minAmount1
);
```

Listing 2.5: src/provider/SmartProvider.sol

Suggestion Remove the redundant code.

2.2.2 Potential zero token minting in function `_mintToMPCs()`

Status Confirmed

Introduced by `Version 1`

Description The contract `SlisBNBxMinter` implements the function `_mintToMPCs()`, which distributes tokens to authorized wallets according to their allocation caps. However, the function uses a less-than-or-equal comparison that permits execution when the current balance equals the wallet cap, creating a logical contradiction where the protocol attempts to mint zero tokens.

```
    if (balance <= wallet.cap) {
      uint256 toMint = balance + leftToMint > wallet.cap ? wallet.cap - balance : leftToMint;
      // mint slisBNBx to the wallet
      SLISBNB_X.mint(wallet.walletAddress, toMint);
      // add up balance
      wallet.balance += toMint;
      // deduct leftToMint
      leftToMint -= toMint;
    }
```

Listing 2.6: src/utils/SlisBNBxMinter.sol

Suggestion Revise the logic accordingly.

2.2.3 Add zero value check in wallet configuration

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `SlisBNBxMinter` implements the function `addMPCWallet()`, which allows the `MANAGER` role to configure new `MPC` wallets with specific allocation `caps`. However, the function does not validate that the `cap` parameter exceeds zero. It is recommended to add a non zero value check on the `cap` parameter.

```
/**
 * @dev Add MPC wallet
 * @param walletAddress - address of the MPC wallet
 * @param cap - cap of the MPC wallet
 */
function addMPCWallet(address walletAddress, uint256 cap) external onlyRole(MANAGER) {
  require(walletAddress != address(0), "zero address provided");
  // check if the wallet already exists
  for (uint256 i = 0; i < mpcWallets.length; ++i) {
    require(mpcWallets[i].walletAddress != walletAddress, "Wallet already exists");
  }
  // add the wallet
  mpcWallets.push(MPCWallet(walletAddress, 0, cap));
  // emit event
  emit MpcWalletAdded(walletAddress, cap);
}
```

Listing 2.7: src/utils/SlisBNBxMinter.sol

Suggestion Add a non zero value check.

2.3 Note

2.3.1 LP token value calculation depends on the function `get_virtual_price()`

Introduced by `Version 1`

Description The contract `SmartProvider` calculates the value of LP tokens through the function `peek()`, which calls `get_virtual_price()` from external DEX contracts. The calculation's accuracy depends on `get_virtual_price()` returning the expected value.

```
function peek(address _token) external view returns (uint256) {
  if (_token == TOKEN || _token == dexLP) {
    // if token is dexLP, return the price of the LP token
    // LP value = min(token0_price, token1_price) * virtual_price
    uint256 minPrice = UtilsLib.min(_peek(token(0)), _peek(token(1)));
    uint256 virtualPrice = IStableSwap(dex).get_virtual_price(); // 1e18
    return (minPrice * virtualPrice) / 1e18;
  }

  return _peek(_token);
}
```

Listing 2.8: src/provider/SmartProvider.sol

Feedback from the project The project team stated that they are using Lista's `StableSwapPool` as the `DEX`, which implements reentrancy protection in the `get_virtual_price()` function to ensure it returns the correct value.

2.3.2 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `MANAGER`) can conduct sensitive operations, which introduces potential centralization risks. For example, `MANAGER` can add a walletAddress to be a `MPC` wallet based on the protocol. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS