Lista DAO: Moolah & CDP PR

Security Review

Cantina Solo review by:
Slowfi, Lead Security Researcher

March 24, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Incorrect caller propagation in `withdrawFor` prevents migration of `BTCB` and `wBETH` positions . . . 4
- 3.2 Low Risk . . . 5
  - 3.2.1 Migration reverts for positions with zero CDP debt . . . 5
- 3.3 Informational . . . 5
  - 3.3.1 Authorization trust model assumption . . . 5
  - 3.3.2 `releaseFor` introduces inconsistent LP accounting assumptions when withdrawing on behalf of users . . . 6
  - 3.3.3 `BNB` migration fallback uses the CDP side `BNB` amount instead of the released `slisBNB` amount . . . 6

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A security review is a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While the review endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that a security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Lista DAO functions as the open-source decentralized stablecoin lending protocol powered by LSDfi.

From Mar 17th to Mar 18th the security researchers conducted a review of moolah and lista-dao-contracts on commit hashes 528802d3 and 15fe97be respectively. A total of **5** issues were identified:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	3	0	3
Total	**5**	**2**	**3**

2.1 Scope

The security review had the following components in scope for moolah and lista-dao-contracts on commit hashes 528802d3 and 15fe97be respectively:

```
├── lista-dao-contracts
│   └── contracts
│       ├── Interaction.sol
│       └── ceros
│           ├── provider
│           │   └── SlisBNBProvider.sol
│           └── upgrades
│               └── HelioProviderV2.sol
└── moolah/src/utils/PositionMigrator.sol
```

3 Findings

3.1 High Risk

3.1.1 Incorrect caller propagation in `withdrawFor` prevents migration of BTCB and wBETH positions

Severity: High Risk

Context: Interaction.sol#L379-L382

Description: The function `withdrawFor` from contract `Interaction` forwards `msg.sender` as the `caller` argument into `_withdraw`.

For collateral types without an associated Helio provider, such as BTCB and wBETH, `_withdraw` enforces that `caller == participant`. However, during the migration flow, `withdrawFor` is invoked by the `PositionMigrator` contract on behalf of the user, so `msg.sender` is the migrator rather than the position owner.

As a result, the direct `Interaction` withdrawal path for BTCB and wBETH always reverts with `Interaction/Caller must be the same address as participant`, making these in-scope collateral types non-migratable through the intended flow.

At the same time, because `withdrawFor` only checks that the caller is the migrator and receives the user address as an input parameter, the function remains callable by the migrator for arbitrary participants, but the internal caller model is still incompatible with direct `Interaction` collaterals. This breaks the migration path specifically for assets that rely on `Interaction` rather than an external provider for collateral release.

Proof of Concept: The following test shows that a BTCB position cannot be migrated because `withdrawFor` propagates the migrator as `caller`, causing `_withdraw` to revert in the direct collateral path:

```
function test_poc_fork_migratorCannotWithdrawDirectInteractionCollateral() public {
    uint256 debtBefore = interaction.borrowed(BTCB, USER_BTCB);
    uint256 collateralBefore = _cdpCollateral(USER_BTCB);

    assertGt(debtBefore, 0, "expected active BTCB debt on the fork");
    assertGt(collateralBefore, 0, "expected active BTCB collateral on the fork");
    assertEq(interaction.helioProviders(BTCB), address(0), "expected direct Interaction
    ↪  collateral");

    vm.prank(USER_BTCB);
    vm.expectRevert(bytes("Interaction/Caller must be the same address as participant"));
    migrator.migratePosition(btcbMigrationParams, false, 0);

    assertEq(interaction.borrowed(BTCB, USER_BTCB), debtBefore, "debt changed despite
    ↪  revert");
    assertEq(_cdpCollateral(USER_BTCB), collateralBefore, "collateral changed despite
    ↪  revert");
    assertEq(IERC20(BTCB).balanceOf(address(migrator)), 0, "migrator should not receive
    ↪  BTCB");
}
```

Recommendation: Consider to adapt `withdrawFor` so that the migration-specific flow does not reuse the regular caller assumptions enforced by `_withdraw` for direct `Interaction` collaterals.

Consider to introduce a dedicated migration withdrawal path, or to pass the effective participant context in a way that remains compatible with BTCB and wBETH withdrawals while preserving the intended access control guarantees.

Lista DAO: Fixed in PR 177.

Reviewer: Fix verified.

3.2 Low Risk

3.2.1 Migration reverts for positions with zero CDP debt

Severity: Low Risk

Context: PositionMigrator.sol#L164

Description: The function `migratePosition` from contract `PositionMigrator` always initiates the migration flow by calling:

```
MOOLAH.flashLoan(LISUSD, cdpDebt, data);
```

where `cdpDebt` is obtained from the user's CDP position.

If a user has zero outstanding debt, `cdpDebt` will be equal to zero. In this case, the migration still attempts to execute a flash loan with a zero amount.

Depending on the implementation of `Moolah.flashLoan`, zero-amount flash loans are typically not supported and may revert. As a result, users with fully repaid CDP positions cannot migrate their collateral using this flow, even though they may still hold collateral in the system.

This creates an unintended restriction where positions without debt cannot be migrated through the provided mechanism.

Recommendation: Consider to handle the zero-debt case explicitly by bypassing the flash loan flow and directly executing the collateral withdrawal and supply steps.

Alternatively, consider to validate that `cdpDebt > 0` before initiating the migration and provide a clear revert reason or separate migration path for zero-debt positions.

Lista DAO: Fixed in PR 150.

Reviewer: Fix verified.

3.3 Informational

3.3.1 Authorization trust model assumption

Severity: Informational

Context: PositionMigrator.sol#L19

Description: The migration flow requires users to authorize the `PositionMigrator` contract via `Moolah.setAuthorization(migrationToolAddr, true)` before executing the migration.

The `PositionMigrator` contract is implemented as an upgradeable UUPS proxy, where upgrades are controlled by an account holding `DEFAULT_ADMIN_ROLE` through the `_authorizeUpgrade` function.

As a result, users grant authorization to a proxy address whose implementation can change over time, rather than to a fixed and immutable contract. This proxy is also set as the privileged migrator in the CDP system, enabling it to repay debt and withdraw collateral on behalf of users.

Since the authorization is tied to the proxy address, it remains valid across future upgrades. Any new implementation introduced by the admin would retain the previously granted permissions without requiring additional user approval.

Recommendation: Consider to clearly communicate this behavior to users, highlighting that the authorization is granted to an upgradeable contract controlled by an administrative role.

Additionally, consider to provide mechanisms to limit the scope or duration of the authorization, or allow users to easily revoke permissions after migration.

Lista DAO: Acknowledged. The `DEFAULT_ADMIN_ROLE` for `PositionMigrator` proxy will be assigned to a `Timelock` contract with a 1-day delay, ensuring that any implementation upgrade is subject to a public waiting period before execution. Additionally, users will be able to revoke their authorization to the migrator directly through the frontend after migration is complete.

Cantina Managed: Acknowledged by Lista DAO team.

3.3.2 `releaseFor` introduces inconsistent LP accounting assumptions when withdrawing on behalf of users

Severity: Informational

Context: SlisBNBProvider.sol#L172-L187

Description: The function `releaseFor` from contract `SlisBNBProvider` enables the migrator to withdraw LP-backed collateral on behalf of a user by calling:

```
_withdrawLp(_account, _amount);
```

This introduces a different execution model compared to the original design of `_withdrawLp`, where the account being withdrawn was assumed to be the one being properly synchronized before any burn or accounting updates.

In the migration flow, `msg.sender` is the migrator, while `_account` represents the user. As a result, the withdrawal logic operates on user balances that may not have been recently synchronized against mutable parameters such as `exchangeRate` and `userLpRate`.

If these parameters have changed since the last user interaction, the burn logic may use updated rates against stale stored balances. This can lead to reverts or inconsistent LP accounting depending on how much the rates have drifted.

Recommendation: Consider to ensure that user LP balances are explicitly synchronized within the migration withdrawal path before applying any burn or accounting logic.

Additionally, consider to avoid relying on external synchronization processes to guarantee correctness of on-chain migration flows.

Lista DAO: Acknowledged. Whenever the `exchangeRate` changes, backend automatically invokes `syncUserLp` for affected users, ensuring that both user LP balances and reserve LP accounting are up to date prior to any migration withdrawal.

Cantina Managed: Acknowledged by **Lista DAO** team.

3.3.3 `BNB` migration fallback uses the CDP side `BNB` amount instead of the released `slisBNB` amount

Severity: Informational

Context: PositionMigrator.sol#L198-L200

Description: The function `onMoolahFlashLoan` from contract `PositionMigrator` handles BNB-backed migrations by releasing CDP collateral through `releaseInTokenFor`, which returns the collateral in the form of `slisBNB`.

However, if the target Moolah market has no collateral provider configured, the fallback branch supplies `data.collateralAmount` as the collateral amount. This value represents the original BNB-side amount from the CDP position, not the actual amount of `slisBNB` received by the migrator after the conversion.

These two amounts are not guaranteed to match. As a result, the fallback branch may attempt to supply more `slisBNB` than the migrator actually holds, causing the migration to revert.

This issue may remain latent while the target `slisBNB` market is configured with the expected collateral provider, since in that case the provider branch is used instead of the fallback path. However, the fallback logic is still inconsistent with the asset amount actually received in the BNB migration flow.

Recommendation: Consider to use the actual `slisBNB` amount released by `releaseInTokenFor` when executing the no-provider fallback branch.

Additionally, consider to avoid reusing the original CDP-side BNB amount after the collateral has already been converted into `slisBNB`.

Lista DAO: Acknowledged. We will ensure that the `slisBNBProvider` is configured for all Moolah markets where the collateral token is `slisBNB`, guaranteeing that the fallback branch is never reached in the `BNB` migration flow.

Cantina Managed: Acknowledged by Lista DAO team.