CANTINA

Seamless leverage tokens

Security Review

Cantina Managed review by:

Denis Miličević, Lead Security Researcher
Om Parikh, Security Researcher

May 8, 2025

Contents

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

Cantina Managed provides 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 Cantina Managed 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 the Cantina Managed 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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Seamless turns leveraged DeFi strategies into simple ERC-20 tokens - easy to access, easy to use.

From Apr 1st to Apr 10th the Cantina team conducted a review of seamless-leverage-tokens on commit hash 4e527768. The team identified a total of **11** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	2	2	0
Gas Optimizations	1	1	0
Informational	6	0	6
Total	**11**	**5**	**6**

3 Findings

3.1 Medium Risk

3.1.1 Incorrect rounding in `_convertToShares` for withdrawal-based actions leads to 1 dead share for users

Severity: Medium Risk

Context: LeverageManager.sol#L349-L362, LeverageManager.sol#L381, LeverageManager.sol#L419

Description: The `_convertToShares` function currently does floor rounding in all cases. This works correctly for any deposit actions. In the case of withdrawals, it ends up rounding down the shares needed to withdraw for a set amount of equity. Thereby, this leads to instances where if users attempt to withdraw all equity they have that requires more than 1 share, such as in the case of 50 shares representing the user's entire equity, they will have 1 share remaining. The final effect is different depending on a combination of decimal and price differential between the associated collateral and debt tokens for a Leveraged Token.

In the ETH2x Long case, the collateral asset is ETH represented with 18 decimals, while the debt asset is USDC represented with 6. If there's a lone user in the LT and withdraws all their equity, they will succeed in withdrawing it and repaying all debt. Following this, all equity is withdrawn, but the user will remain with 1 dead share. Even if more equity enters, this dead share is unusable in this case, and can't be withdrawn.

In the ETH2x Short case, the collateral is USDC and ETH is the debt. Since ETH as the debt has more precision, it leads to a case where requesting all equity, results in 1 leftover share, with fractional equity and debt being represented in that share. A secondary withdrawal action is required with 1 share to completely remove a user's position. This case doesn't lead to dead shares, but requires 2 withdrawal actions.

Other than the dead share that can't be withdrawn or leftover share requiring 2 withdrawals, the effect this can have on Leveraged Tokens in the first case is an artificial overinflation of the total supply (as these shares will permanently keep it increased even though there's no equity backing it) that can't be removed. This results in an increased dilution to current Leveraged Token holders proportional to the amount of unique accounts that have attempted a complete withdrawal and have 1 dead share remaining, to the benefit of future deposits. This effect should generally not be significant enough in non-inflation attacked tokens, however, this effect can be greatly amplified if a token has been inflated.

Recommendation: `_convertToShares` should take an additional parameter, to stay consistent, likely a `ExternalAction` typed parameter, which adjusts the rounding based off the action type. `Math.Rounding.Floor` for deposit derived actions, and `Math.Rounding.Ceil` for withdrawals.

An example:

```
- function _convertToShares(ILeverageToken token, uint256 equityInCollateralAsset)
+ function _convertToShares(ILeverageToken token, uint256 equityInCollateralAsset, ExternalAction action)
      internal
      view
      returns (uint256 shares)
  {
      ILendingAdapter lendingAdapter = getLeverageTokenLendingAdapter(token);

      return Math.mulDiv(
          equityInCollateralAsset,
          token.totalSupply() + 10 ** DECIMALS_OFFSET,
          lendingAdapter.getEquityInCollateralAsset() + 1,
-         Math.Rounding.Floor
+         action == ExternalAction.Deposit ? Math.Rounding.Floor : Math.Rounding.Ceil
      );
  }
```

an appropriately updating all relevant calls with the action in context being passed.

Seamless: Fixed in commit da5b6f67 following the recommendation by the auditor; share calculation rounds up on deposit and rounds down on withdraw.

Cantina Managed: Fix verified.

3.1.2 Protect external facing `LeverageManager` functions from reentrancy

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: A number of the external functions in `LeverageManager`, namely `deposit`, `withdraw`, `rebalance`, and `createNewLeverageToken` read and change state, while doing external calls to user-specifiable contracts. This could result in exploits by temporarily affecting the contract's state through one of these functions initial calls and then re-entering into another to take advantage of this modified state.

Some attacks have been considered with `deposit` and `withdraw` as the entrypoints, but utilizing reentrancy on them would likely not benefit an attacker. Others utilizing `rebalance` may allow a Leveraged Token to be put into a temporarily invalid state, where `isStateAfterRebalanceValid` would normally not pass, however, a reentrance would allow this invalid state to be utilized for minting shares at a discounted rate. Following the minting, the middle execution of rebalance can be cleaned up and it reset to a valid state to allow the transaction to confirm without reverting.

Recommendation: Protect the noted functions with the `nonReentrant` modifier to disallow any potential for reentrancy and attacks it may yield. This should also be applied to any other external facing stateful functions.

Seamless: Fixed in commit db971763 following the recommendation by the auditor; nonReentrant modifiers have been added to LeverageManager.createNewLeverageToken, LeverageManager.deposit, LeverageManager.withdraw, and LeverageManager.rebalance.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Remove configurability of `DECIMALS_OFFSET` parameter or account for offset in `LeverageToken` contract

Severity: Low Risk

Context: LeverageManager.sol#L65, LeverageManager.sol#L358, LeverageToken.sol#L20

Description: The `LeverageManager` contract utilizes a `DECIMALS_OFFSET` parameter, which is intended to provide a degree of inflation attack equal to its value. The constant is currently set to 0, which is the default value, and which in this contract's case provides no inflation attack protection.

If it is configured to a non-zero value prior to deployment, the `LeverageToken` is not passed this value to appropriately add the offset its `decimals()` function as should be done. The offset is intended to create virtual shares, and without including this offset in `decimals()` it can lead to situations where virtual shares end up exceeding a full Leverage Token unit representation, in the case of an offset of 18 or higher. This in practice would mostly be a visual bug.

Recommendation: 3 mutually exclusive options have been represented to the client, with option #2 being favoured as the audit was completed with that arithmetic in place.

1. Appropriately implement decimals offset to also affect the leverage token, and set it to some amount you wish to act as the cost to an inflation attack. This addition of cost to the attack is not necessary, as appropriate use of min/max shares in `deposit` and `withdraw` offers sufficient slippage protection to protect users from a full or significant value stealing inflation attack, assuming appropriate setting of those parameters.
2. Remove the potential for configuration, even as a constant, where its value could be reconfigured prior to deploy, as changing the value from its current default is unsupported without the implementation of point 1. Continue using the arithmetic as is, it could be noted in the NatSpec that the shares calculation is based off the latest OZ vault, utilizing the default 0 decimal offset value, which is one. This does introduce a slightly increased imprecision that is more in favor of the contracts, but avoids the necessity for different arithmetic on an empty vault base case.
3. Re-implement the arithmetic with separate logic for base cases, and more precise arithmetic usable in non-base cases.

Seamless: Fixed in commit da5b6f67 following the third recommendation by the auditors; the arithmetic has been re-implemented with separate logic for base cases, which will result in more precision.

Cantina Managed: Fix verified.

3.2.2 Fractional collateral remainder from a rebalance could be pulled with lone `removeCollateral` action

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The tokens utilized in Leveraged Tokens will have a variable pricing relationship between one another. Due to this, there can be instances of fractional collateral being left on the table by one rebalancer when they repay debt during a rebalance. Another rebalancer could then claim this collateral, by calling `removeCollateral` without any input tokens or other actions.

The remainder factor in the case of ETH2x Long, where the collateral asset's price is a multiple of the debt asset, is positively affected by the decimal differential and inversely by the collateral price in debt. The maximal remainder amount can be calculated via.

```
(Collateral Base Unit) / ((Collateral Price in Debt) * (Debt Base Unit)) - 1
```

or in the case of $1,300 ETH.

```
1e18 / (1300 * 1e6) - 1
```

resulting in a floored value of 769,230,768 wei. To give a more specific example using the above formulation:

- ETH2x long, price starts at $2,000 then drops to $1,300 entering a preliquidation state.
- A rebalancer provides 1 unit of USDC as repay and removes 1 wei unit of WETH collateral.
- A subsequent user can call rebalance with no input tokens, and just do a `removeCollateral` action for 769,230,768 wei units of WETH, removing this fractional equity from token holders, albeit the prior rebalancer was technically due this entire amount anyways.

Recommendation: Consider disallowing withdrawal of collateral, that is not accompanied by some repay action. The best way to ensure this is to extend pre liquidation checks to require improvement of the collateral ratio towards its target, rather than just accepting a non-change as well.

Seamless: Fixed in commit 3dc8edd9 following the recommendation by the auditor by checking in the rebalance adapter that rebalances must result in a collateral ratio that is better than before rebalance.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 Struct that is written to and accessed in storage could be optimized from 3 slots to 1

Severity: Gas Optimization

Context: DataTypes.sol#L38-L46

Description: The `Auction` struct defined in `DataTypes.sol` has 3 properties, 1 of which is `bool`, and the other 2 are `uint256`. A `bool` already takes up the smallest amount of bitspace and is essentially a `uint8`. The noted `uint256` properties are timestamps. A `uint48` representation is sufficient for unix time millions of years into the future.

Recommendation: Since this specific struct writes and accesses storage, some significant storage gas savings could be saved here by cutting the access from 3 slots to 1 slot, especially when writing. The timestamp properties should be set to `uint120` such that they tightly pack together with each other and the `bool` property to 1 slot, and are consistent in size with one another.

Seamless: Fixed in commit ec4d99c9.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 `createNewLeverageToken` is permissionless and accepts any adapters, including possibly malicious

Severity: Informational

Context: LeverageManager.sol#L162, LeverageManager.sol#L176-L179

Description: The `LeverageManager` lacks enforcement of adapters being used by Leveraged Tokens when they are created via `createNewLeverageToken`. This means that adapters outside of this audit could be used which could be unsafe or completely malicious on purpose. Users must not assume that just because one Leveraged Token launched from a specific `LeverageManager` is safe, that its guarantees or safety is transferred to a different one. Adapters handle a brunt of the logic with regards to liquidation and lending protocol interactions, and ultimately hold the collateral. Therefore this can lead to serious risk if unaudited or unreviewed adapters are behind a Leverage Token.

This has been noted as a purposeful design decision as it's intended to be permissionless and flexible.

Recommendation: It has been recommended to consider implementing a whitelist on the contract level of adapters with specific `extcodehash` output that are audited or official. The list could be extended by the governing DAO.

If that is considered to be too restrictive, at least filter for official adapter tokens on the UI side, however, note that this extends the attack surface to the UI as well, where malicious tokens are launched from the `LeverageManager` and if the UI is compromised and lists these tokens, users can lose funds.

In all cases, users should ideally inspect each Leveraged Token themselves and ensuring the adapters used are either official or not malicious, hopefully backed by a list of known safe adapters that can be derived just from a hash.

Seamless: Acknowledged. Users should inspect and understand the risk of using any particular LeverageToken, which by design can be created permissionlessly with arbitrary rebalance adapters, lending adapters, and tokens. This behaviour is expected and documented.

Cantina Managed: Acknowledged.

3.4.2 `MorphoLendingAdapter` is susceptible to oracle collusion or manipulation

Severity: Informational

Context: MorphoLendingAdapter.sol#L104-L121

Description: Much of the critical logic in the lending adapter is dependent on accurate data from an oracle. This means oracle risk is extended to this series of contracts, exposing them to potential oracle collusion, manipulation, or failure. This dependency is a fundamental part of the design needing lending protocols which in turn require these oracles fundamentally themselves.

A compromised oracle has the ability to set Leveraged Tokens in to compromised states that could result in user fund loss.

Recommendation: In the case of this Lending Adapter which uses Morpho as its lending protocol, Morpho allows for custom selection of the oracle for a market. Users should follow the same recommendations for the Leveraged Tokens utilized Morpho market, as Morpho recommends, in that they verify the used oracle implementation, understand the price sources being used and consider potential manipulation vectors or failure modes.

Seamless: Acknowledged. Users should inspect and understand the risk of using any particular LeverageToken, which by design can be created permissionlessly with arbitrary lending adapters and thus arbitrary oracles. This is inclusive of the oracle used by the underlying lending protocol (and thus configured lending adapter) used by a LeverageToken. This behaviour is documented.

Cantina Managed: Acknowledged.

3.4.3 Rebalancers must correctly specify `tokensOut` when rebalancing or lose tokens to others

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Rebalancers are required to provide an exact correct amount when calling `LeverageManager.rebalance` for the `tokensOut` parameter, which represents the expected withdrawable amounts resulting from their actions, or `tokensIn` excesses.

In a vacuum, this appears to be a simple expectation, however, under real network conditions, the market and its parameters is dynamic and prone to changing prior to the pending transaction being confirmed. In the ideal case, the rebalancer ends up overspecifying `tokensOut`, which would revert the entire transaction. But, if it's underspecified, that rebalancer would only receive a suboptimal amount. Then, any other rebalancer, even of another token could pull that remainder out by specifying that token and amount in their `tokensOut` argument.

Recommendation: As the `LeverageManager` is not intended to actually hold tokens beyond a transaction's interaction, consider transferring the balance in its entirety, for any tokens noted in `tokensOut` that are specified. This would avoid the suboptimal scenario which could appear.

If additionally, any `tokensIn` tokens that still have a balance are also considered here, it would avoid a potential issue where excesses of `tokensIn` are not accounted for in `tokensOut`.

If the above are not implemented, rebalancers could and should use their own router contract that would read current contract values and appropriately update the parameters so they are correct and current and ensure that the involved token's balances did not increase in the `LeverageManager` or attempt a withdrawal of them.

Seamless: Acknowledged. Our stance is that any mistake in the amount of tokens to receive for rebalancing a LeverageToken is on the onus of the rebalancer. Also, it is generally expected that rebalancers will use their own router contract, as mentioned in the auditor recommendation.

Cantina Managed: Acknowledged.

3.4.4 General notes on morpho markets integration

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Due to mechanism how leverage token functions, there can be several morpho markets which are best to avoid due to compatibility or add with extreme precaution. Following is a non-exhaustive list of such markets compostion which has loan token or collateral token that:

- Is metamorpho share, because the leverage would be non-linear due to nested exposure and calculations might not work.
- Is ERC4626 whose oracle uses `convertToShares` and/or `convertToAssets` which is pricing assets on chain, because leverage manager is not re-entrancy safe and can be executed in context of `onMorphoXYZ` callbacks and might change `.price()` value (read-only re-entrancy).
- Is ERC777, due to re-entrancy risk.
- Is high-volatity and low-precision pair (consider something like PEPE/WBTC market for e.g whose pricing can be fitted in `1e36` oracle scale but such accurate pricing is not available in first place).

Recommendation: No code changes required, following should be documented and taken care of during day to day operations.

Seamless: Acknowledged. `LeverageTokens` inherit the constraints of the underlying lending protocol it uses, so if there is some recommendations/restrictions from the underlying protocol to not launch a certain market, `LeverageTokens` would inherit those recommendations/restrictions by design.

Cantina Managed: Acknowledged.

3.4.5 General notes on sequencer uptime

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The leverage token contracts will be deployed on base which is currently operated by a single sequencer. In case of sequencer down-time, following scenarios can happen which might damage or extract value from protocol:

- If the price movement was large and in unfavourable direction then it can lead to liquidation before rebalance on morpho.
- If there are no sequencer uptime checks in oracle, actions (deposit, withdraw, borrow, repay, supply) can be settled at stale prices.
- Value extraction upto certain extent from the DEX (uniswap and aerodrome) swaps on transactions via periphery.
- Any upstream contracts integrating leverage tokens might not be able to fairly price the value of leverage token until operations are normal and can lead to unwanted secondary effects.

Recommendation: It should be document or added in guidelines and impact on some of the above issues might be reduced by implementing offchain monitoring solutions.

Seamless: Acknowledged. Any implementation or mechanism to handle sequencer down time must be implemented at the lending protocol, it should not be implemented by Leverage Tokens.

Cantina Managed: Acknowledged.

3.4.6 Temporary DoS due to liquidity caps on lending adapters

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: For various other lending protocols which have caps on collateral token deposits (excluding morpho, since morpho doesn't have caps on supplyCollateral), if deposit cap is reached, adding collateral will fail. Same way, borrow fails when there is no liquidity available to borrow (including morpho). Following are worst case consequences of this:

- Deposits will revert.
- Rebalancing down will revert.

Impact:

- Any protocol building on top of seamless leverage tokens might face DoS.
- Actual position will be inconsistent with what is demanded by strategy (i.e instead of 2x leverage, it will be at 1.8x) changing profile of returns generated.

No loan token liquidity being available to borrow is not so uncommon in practice, especially during higher volatility periods. Even if the borrow APY spikes up to max, it can still take substantial and non-certain duration for capacity to be available again.

However, in Aave v3.x and Euler (depends how Euler is integrated) where the collateral supplied is loaned out to someone else markets are not isolated impact is more severe, collateral token withdraw can fail, extending failure to withdraw and rebalancing up.

Recommendation: Due to limitations of the downstream lending protocols, it might be complex to completely mitigate this problem. However, It should be documented and outlined in risk management strategy. Also, available liquidity on concerned protocols and markets should be monitored offchain to trigger an alert if caps are closer to getting filled or any sudden changes.

Seamless: Acknowledged. In general, we intentionally want the lending pool's existing mechanisms—such as deposit caps, interest rate spikes, or liquidity incentives—to regulate how quickly liquidity returns. If a deposit reverts because a pool is capped, that indicates the pool is imposing the correct risk safeguards. This is documented.

For example, for Leverage Tokens using a `MorphoLendingAdapter`, it's possible for rebalance down (add collateral, borrow debt to decrease collateral ratio) to revert due to the utilization of the underlying Morpho market restricting borrows until more liquidity is supplied.

For future lending adapters (beyond Morpho), developers should consider the implications of the specific behaviour of that lending pool to write a correct adapter.

Cantina Managed: Acknowledged.