Revert Lending Protocol

Security Review

Cantina Managed review by:

0xWeiss, Security Researcher
Víctor Martínez, Security Researcher

November 3, 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 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

Revert develops analytics and management tools for liquidity providers in AMM protocols.

From Sep 12th to Oct 7th the Cantina team conducted a review of lend on commit hash f4190ae9. The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	4	3	1
Gas Optimizations	0	0	0
Informational	5	5	0
Total	**10**	**9**	**1**

3 Findings

3.1 Critical Risk

3.1.1 Incorrect onERC721Received Logic Causes Staked Positions to be Undetectable, Breaking Liquidation & NFT Return Flows

Severity: Critical Risk

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

Description: The vault's `onERC721Received` function cannot distinguish between:

- NFTs deposited as collateral.
- NFTs returned back to the vault due to unstaking.

Because the vault handler only checks:

```
if (oldTokenId == 0) {
    loans[tokenId] = Loan(0);
    _addTokenToOwner(owner, tokenId);
}
```

which assumes every inbound NFT is a new deposit, wiping loan state and assigning ownership incorrectly when an unstake event occurs.

This causes the following impacts:

- Staked positions become unliquidatable.
- Users can permanently lose their NFTs when unstaking through the vault.

Recommendation: Re-engineer the `onERC721Received` function so that it can identify when an NFT is first deposited or it comes from an unstaked position. This can be checked by checking if the owner of such NFT is not `address(0)`. If the NFT comes from the `GaugeManager` when being unstaked, then do not white the `Loan` data as it will be needed for functions like liquidation. An example on how to accomplish this:

```
  if (oldTokenId == 0) {

+   if (tokenOwner[tokenId] == address(0)) {

            address owner = from;
             if (data.length != 0) {
                 owner = abi.decode(data, (address));
             }
             loans[tokenId] = Loan(0);



             _addTokenToOwner(owner, tokenId);
             emit Add(tokenId, owner, 0);
+     } else {
+                 // EXISTING token returning from staking/gauge
+                 // Do nothing - loan and ownership data are already correct
+                 // The NFT is just being returned to vault custody
+         }
         } else {
```

Revert Finance: Fixed in commit 894a305a.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Gauge address is not properly verified

Severity: Low Risk

Context: GaugeManager.sol#L77

Description: The `setGauge` setter should add gauge address verification so that the gauge address actually is the proper one according to: CLPool.sol#L47

```
function setGauge(address pool, address gauge) external onlyOwner {
   poolToGauge[pool] = gauge;
}
```

Recommendation: Add the following validation:

```
  function setGauge(address pool, address gauge) external onlyOwner {
+     require(IAerodromeSlipstreamPool(pool).gauge() == gauge, "wrong gauge");
      poolToGauge[pool] = gauge;
  }
```

Revert Finance: Fixed in commit 894a305a.

Cantina Managed: Fix verified.

3.2.2 Missing validation of max `aeroSplitBps`

Severity: Low Risk

Context: GaugeManager.sol#L212

Description: The `compoundRewards` and `executeV3UtilsWithOptionalCompound` functions miss validation so that `aeroSplitBps` does not surpass 10_000 basis points:

```
if (swapData0.length > 0) {
    (, amount0) = _routerSwap(
        RouterSwapParams(
            aeroToken,
            IERC20(token0),
            (aeroAmount * aeroSplitBps) / 10000, // <<<
            minAmount0,
            swapData0
        )
    );
}
```

Recommendation: Add the following requirement to both, the `compoundRewards` and `executeV3UtilsWithOptionalCompound` functions:

```
  function compoundRewards(
        uint256 tokenId,
        bytes calldata swapData0, bytes calldata swapData1, uint256 minAmount0,
        uint256 minAmount1, uint256 aeroSplitBps, uint256 deadline
    ) external nonReentrant {
        address owner = positionOwners[tokenId];
        bool fromVault = isVaultPosition[tokenId];
        require(
            msg.sender == owner ||
            (fromVault && msg.sender == address(vault)),
            "Not authorized"
        );
+       require( aeroSplitBps <= 10000, "wrong aeroSplitBps");

        address gauge = tokenIdToGauge[tokenId];
        require(gauge != address(0), "Not staked");
  }
```

```
  function executeV3UtilsWithOptionalCompound(
         uint256 tokenId,
         V3Utils.Instructions memory instructions,
         bool shouldCompound,
         bytes memory aeroSwapData0,
         bytes memory aeroSwapData1,
         uint256 minAeroAmount0,
         uint256 minAeroAmount1,
         uint256 aeroSplitBps
     ) public nonReentrant returns (uint256 newTokenId) {
         require(v3Utils != address(0), "V3Utils not configured");
+        require(aeroSplitBps <= 10000, "Invalid split");
```

Revert Finance: Fixed in commit 894a305a.

Cantina Managed: Fix verified.

3.2.3 Missing input validation in the `create` function

Severity: Low Risk

Context: V3Vault.sol#L355

Description: Several parts of the code currently lack proper validation. Notably, most constructors and multiple functions in the `V3Vault` contract, such as `create` and `createWithPermit`, do not perform necessary checks. For example:

```
function create(uint256 tokenId, address recipient) external override {
       nonfungiblePositionManager.safeTransferFrom(msg.sender, address(this), tokenId, abi.encode(recipient));
   }
```

Recommendation: Implement validation in all constructors and critical functions, including `create`. For instance, ensure that the recipient is a valid address:

```
  function create(uint256 tokenId, address recipient) external override {
+     require(recipient != address(0), "incorrect address");
      nonfungiblePositionManager.safeTransferFrom(msg.sender, address(this), tokenId, abi.encode(recipient));
  }
```

Revert Finance: Fixed in commit 62eecc71.

Cantina Managed: Fix verified.

3.2.4 `totalAssets` does not follow the ERC-4626 standard specification

Severity: Low Risk

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

Description: The `totalAssets` function does not comply with the ERC4626 standard specification. According to ERC4626, `totalAssets` MUST include any compounding that occurs from yield and represent the total amount of underlying assets managed by the vault.

Current Implementation:

```
function totalAssets() public view override returns (uint256) {
    return IERC20(asset).balanceOf(address(this));
}
```

This implementation only returns the idle balance held in the vault contract, completely ignoring the outstanding debt (assets lent out to borrowers) that is actively earning interest.

The vault operates as a lending protocol where:

1. Lenders deposit assets and receive shares.
2. Assets are lent to borrowers who provide NFT collateral.
3. Borrowers pay interest on their debt.
4. Interest accrues over time, increasing the total value managed by the vault.

The true total assets should be: `balance + outstandingDebt`, where outstanding debt includes accrued interest. The internal `_getBalanceAndReserves` function (lines 956-967) correctly calculates this:

```
function _getBalanceAndReserves(uint256 debtExchangeRateX96, uint256 lendExchangeRateX96)
    internal
    view
    returns (uint256 balance, uint256 reserves)
{
    balance = totalAssets(); // Only gets vault balance
    uint256 debt = _convertToAssets(debtSharesTotal, debtExchangeRateX96, Math.Rounding.Up);
    uint256 lent = _convertToAssets(totalSupply(), lendExchangeRateX96, Math.Rounding.Up);
    unchecked {
        reserves = balance + debt > lent ? balance + debt - lent : 0;
    }
}
```

The function correctly recognizes that total value is `balance + debt`, but `totalAssets()` doesn't reflect this.

Impact:

1. ERC4626 Non-Compliance: External integrators expecting ERC4626 compliance will receive incorrect data.
2. Misleading Information: Off-chain systems, analytics tools, and UIs relying on `totalAssets` will underreport vault TVL.
3. Incorrect Pricing: While internal accounting uses proper exchange rates, external callers of `totalAssets` see only a fraction of actual value.
4. Integration Failures: Protocols integrating with this vault as an ERC4626 vault will make incorrect calculations.

Recommendation: Update `totalAssets()` to include both idle balance and outstanding debt with accrued interest:

```
  function totalAssets() public view override returns (uint256) {
-     return IERC20(asset).balanceOf(address(this));
+     (uint256 debtExchangeRateX96,) = _calculateGlobalInterest();
+     uint256 balance = IERC20(asset).balanceOf(address(this));
+     uint256 debt = _convertToAssets(debtSharesTotal, debtExchangeRateX96, Math.Rounding.Up);
+     return balance + debt;
  }
```

Update `_getBalanceAndReserves()` to avoid double-counting:

```
  function _getBalanceAndReserves(uint256 debtExchangeRateX96, uint256 lendExchangeRateX96)
      internal
      view
      returns (uint256 balance, uint256 reserves)
  {
-     balance = totalAssets();
+     balance = IERC20(asset).balanceOf(address(this)); // Direct balance query
      uint256 debt = _convertToAssets(debtSharesTotal, debtExchangeRateX96, Math.Rounding.Up);
      uint256 lent = _convertToAssets(totalSupply(), lendExchangeRateX96, Math.Rounding.Up);
      unchecked {
          reserves = balance + debt > lent ? balance + debt - lent : 0;
      }
  }
```

This ensures `totalAssets()` accurately represents all assets under management, including lent-out assets with accrued interest, bringing the implementation into compliance with the ERC4626 standard.

Revert Finance: Acknowledged. Won't Fix

We agree that including lent-out assets with accrued interest in totalAssets() would better align with ERC4626 semantics and external integrator expectations.

However, we're maintaining the current implementation for consistency with our already-deployed Uniswap V3 vaults. Changing this would create inconsistency across vault versions and break existing integrations.

For integrators needing total managed assets (idle + lent out), the info() function provides both balance and debt

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Unnecessary approval in `migrateToVault`

Severity: Informational

Context: GaugeManager.sol#L563

Description: The `migrateToVault` function has an unnecessary approval to the vault as `safeTransferFrom` has a ”push” mechanism used to transfer nfts directly from this contract to the vault, without the need of a previous approval:

```
// Approve vault to take the NFT
nonfungiblePositionManager.approve(address(vault), tokenId);

// Transfer to vault using safeTransferFrom with recipient encoded
nonfungiblePositionManager.safeTransferFrom(
    address(this),
    address(vault),
    tokenId,
    abi.encode(recipient)
);
```

Recommendation: Remove the superfluous approval:

```
-  // Approve vault to take the NFT
-  nonfungiblePositionManager.approve(address(vault), tokenId);

  // Transfer to vault using safeTransferFrom with recipient encoded
  nonfungiblePositionManager.safeTransferFrom(
      address(this),
      address(vault),
      tokenId,
      abi.encode(recipient)
  );
```

Revert Finance: Fixed in commit 5d25f059.

Cantina Managed: Fix verified.

3.3.2 Stale approval system

Severity: Informational

Context: GaugeManager.sol#L240, GaugeManager.sol#L438, GaugeManager.sol#L511-L518, Swapper.sol#L105

Description: The `GaugeManager` and `Swapper` contracts currently handles ERC20 approvals using the following pattern:

```
if (params.amount0 > 0) {
    IERC20(token0).safeApprove(v3Utils, 0);
    IERC20(token0).safeIncreaseAllowance(v3Utils, params.amount0);
}
if (params.amount1 > 0) {
    IERC20(token1).safeApprove(v3Utils, 0);
    IERC20(token1).safeIncreaseAllowance(v3Utils, params.amount1);
}
```

This is a legacy approach used to support tokens like USDT, which they revert when attempting to change a non-zero allowance. While functional, this approach is considered stale and inefficient as it performs redundant approve(0) and approve(amount) calls.

Recommendation: Replace the approval pattern with OpenZeppelin's forceApprove function.

- `GaugeManager` contract:

```
    if (params.amount0 > 0) {
-               IERC20(token0).safeApprove(v3Utils, 0);
-               IERC20(token0).safeIncreaseAllowance(v3Utils, params.amount0);
+             IERC20(token0).forceApprove(v3Utils, params.amount0);
    }
    if (params.amount1 > 0) {
-               IERC20(token1).safeApprove(v3Utils, 0);
-               IERC20(token1).safeIncreaseAllowance(v3Utils, params.amount1);
+              IERC20(token1).forceApprove(v3Utils, params.amount1);
    }
```

- `Swapper` contract:

```
-  SafeERC20.safeIncreaseAllowance(params.tokenIn, zeroxAllowanceHolder, params.amountIn);
+  SafeERC20.forceApprove(params.tokenIn, zeroxAllowanceHolder, params.amountIn);
```

```
    (bool success,) = zeroxAllowanceHolder.call(params.swapData);
    if (!success) {
        revert SwapFailed();
    }
-   SafeERC20.safeApprove(params.tokenIn, zeroxAllowanceHolder, 0);
+   SafeERC20.forceApprove(params.tokenIn, zeroxAllowanceHolder, 0);
```

Revert Finance: Fixed in commit e4b10f69.

Cantina Managed: Fix verified. Additional fix for swapper merged at commit c79046f9.

3.3.3 `_addTokenToOwner` should emit the `Add` event

Severity: Informational

Context: V3Vault.sol#L389

Description: The Add event is emitted in two different places within the `onERC721Received` function:

1. When `oldTokenId == 0` (new deposit scenario):

```
    if (oldTokenId == 0) {
        address owner = from;
        if (data.length != 0) {
            owner = abi.decode(data, (address));
        }
        loans[tokenId] = Loan(0);

        _addTokenToOwner(owner, tokenId);

        emit Add(tokenId, owner, 0); // <<<
```

2. When `tokenId != oldTokenId` (transformation scenario):

```
    } else {
        if (tokenId != oldTokenId) {
            address owner = tokenOwner[oldTokenId];

            transformedTokenId = tokenId;

            uint256 debtShares = loans[oldTokenId].debtShares;

            loans[tokenId] = Loan(debtShares);

            _addTokenToOwner(owner, tokenId);
            emit Add(tokenId, owner, oldTokenId); // <<<
```

This duplication can be eliminated by having `_addTokenToOwner` emit the event, while accepting the `oldTokenId` as a parameter to preserve the correct context.

Recommendation: Modify the `_addTokenToOwner` function to accept an `oldTokenId` parameter and emit the event:

```
-   function _addTokenToOwner(address to, uint256 tokenId) internal {
+   function _addTokenToOwner(address to, uint256 tokenId, uint256 oldTokenId) internal {
        ownedTokensIndex[tokenId] = ownedTokens[to].length;
        ownedTokens[to].push(tokenId);
        tokenOwner[tokenId] = to;
+       emit Add(tokenId, to, oldTokenId);
    }
```

Update the call sites to pass the appropriate `oldTokenId`:

```
    if (oldTokenId == 0) {
        address owner = from;
        if (data.length != 0) {
            owner = abi.decode(data, (address));
        }
        loans[tokenId] = Loan(0);
-       _addTokenToOwner(owner, tokenId);
-       emit Add(tokenId, owner, 0);
+       _addTokenToOwner(owner, tokenId, 0);
    } else {
        if (tokenId != oldTokenId) {
```

```
            address owner = tokenOwner[oldTokenId];
            transformedTokenId = tokenId;
            uint256 debtShares = loans[oldTokenId].debtShares;
            loans[tokenId] = Loan(debtShares);
-           _addTokenToOwner(owner, tokenId);
-           emit Add(tokenId, owner, oldTokenId);
+           _addTokenToOwner(owner, tokenId, oldTokenId);
```

Revert Finance: Fixed in commit 5cf98f26.

Cantina Managed: Fix verified.

3.3.4 Unused code across the repository

Severity: Informational

Context: Constants.sol#L4

Description: The repository contains several instances of unused code that could be removed to improve readability and maintainability:

In `Constants.sol`:

```
error InterestNotUpdated();

error DebtChanged();

error InvalidTickSpacing();
error AlreadyStaked();
error NotStaked();
error RewardClaimFailed();
```

In `AutoCompound.sol`:

```
import "v3-periphery/libraries/LiquidityAmounts.sol";
```

In `GaugeManager.sol`:

```
import "./utils/Constants.sol";
```

In `V3Vault.sol`:

```
import "v3-core/interfaces/IUniswapV3Pool.sol";
```

Recommendation: These lines do not contribute to the functionality of the contracts and can be safely removed.

Revert Finance: Fixed in commit 62eecc71.

Cantina Managed: Fix verified.

3.3.5 Incorrect PUSH pattern used when collecting staking rewards

Severity: Informational

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

Description: Currently, when unstaking rewards, which is a function called in different occasions, one of them being liquidations, the rewards from the position are claimed and directly sent to the user:

```
uint256 aeroBefore = aeroToken.balanceOf(address(this));

IGauge(gauge).getReward(tokenId);
uint256 aeroAmount = aeroToken.balanceOf(address(this)) - aeroBefore;

if (aeroAmount > 0) {
    aeroToken.safeTransfer(owner, aeroAmount);
}

// Unstake
IGauge(gauge).withdraw(tokenId);
```

Now in case it would be a liquidation and there would be a blacklist function inside the aeroToken contract that blocks transfers to certain addresses or it would be ETH, the current pattern of pushing the rewards to the user would allow for a full DOS on liquidations. The best is always to use PULL over PUSH pattern where the user can claim in a specific function the rewards accumulated so they cant DOS in any way.

Recommendation: Use a PULL over PUSH method where the user can claim their rewards in a separate function.

Revert Finance: Fixed in commit 256d11fe.

Cantina Managed: Fix verified.