Lorenzo OTF contract

Security Review

Cantina Managed review by:

Slowfi, Security Researcher
Xposed, Associate Security Researcher

July 17, 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

Lorenzo is a protocol that enhances Bitcoin liquidity while securing the decentralized world via Babylon.

From Jun 28th to Jul 2nd the Cantina team conducted a review of lorenzo-OTF-contract on commit hash 32860aa4. The team identified a total of **20** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	4	4	0
Medium Risk	4	3	1
Low Risk	5	2	3
Gas Optimizations	2	1	1
Informational	5	1	4
Total	**20**	**11**	**9**

3 Findings

3.1 High Risk

3.1.1 A Malicious User Can Disrupt The Intended Behavior Of The Protocol Through Front Running

Severity: High Risk

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

Description: Users can exploit front running opportunities around the `setUnitNav()` call to guarantee a profit. By monitoring the pending NAV update transaction and depositing just before the NAV increases, a user can obtain USD1+ shares at a lower NAV and then redeem them at a higher NAV within the same settlement period. Because the withdrawal amount is fixed at the moment `requestWithdraw()` is called, the user locks in the profit regardless of subsequent NAV changes.

This undermines the protocol's intended economics, distorts NAV accuracy, and, if more users become aware of this strategy and start doing the same, the protocol may become unsustainable.

Recommendation: Introduce a short lockout period before the end of each settlement cycle during which deposits, redemptions, and withdrawals are temporarily paused. To minimize user disruption, this period can be scheduled during hours of low user activity. This mechanism gives the system time to finalize NAV updates without exposure to front running, helps ensure fair NAV calculation.

Lorenzo: Fixed on commit aa08aa62.

Cantina Managed: Fix verified. The `withdraw()` function now includes logic to check that the withdrawal time must be greater than two period after the withdrawal request.

3.1.2 Swap functions bypass blacklist in `sUSD1PlusVault`

Severity: High Risk

Context: sUSD1PlusVault.sol#L193-L213, USD1PlusVault.sol#L171-L178

Description: The `sUSD1PlusVault` contract inherits blacklist enforcement by overriding the public `transfer` and `transferFrom` functions from `Vault.sol`. However, its `swapToSusd1` and `swapToUsd1Plus` helper functions internally invoke `_transfer` directly, circumventing those public hooks. As a result, a blacklisted address can still move its LP shares to another account by calling one of these swap functions. This undermines the intended blacklist protection, allowing prohibited users to evade restrictions and transfer shares despite being blacklisted.

Remediation: Integrate blacklist checks into the swap pathways. Either replace direct calls to `_transfer` with calls to the public `transfer`/`transferFrom` methods (so that blacklist enforcement triggers), or add explicit blacklist validation before invoking `_transfer` in each swap function. This ensures that blacklisted accounts remain unable to transfer shares through any contract method.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Partially fixed. Now a blacklisted user can not swap `sUSD1PlusVault` shares for `USD1PlusVault` tokens if its whitelisted as `swapToUsd1Plus` contains the modifier `notBlacklisted`. However its possible for a blacklisted user that has `USD1PlusVault` shares to transfer it to another account and redeem them. Also even if blacklisted it is still possible to redeem them, although the account would get stuck with the `sUSD1PlusVault` shares. Nontheless its also important to understand that `USD1PlusVault` contract has also a frozen mechanism that can help to stop suspicious addresses.

3.1.3 Swap functions bypass freeze control in `sUSD1PlusVault`

Severity: High Risk

Context: sUSD1PlusVault.sol#L193-L213, USD1PlusVault.sol#L171-L178

Description: The `sUSD1PlusVault` contract leverages a freeze mechanism in `Vault.sol` to prevent movement of LP shares when flagged as suspicious. This is enforced by overriding the public `transfer` and `transferFrom` functions to check for frozen balances before allowing a transfer. However, the `swapToSusd1` and `swapToUsd1Plus` helper functions invoke the internal `_transfer` method directly, skipping these

public checks. Consequently, even if an address's shares are frozen, it can still transfer them to another account via the swap functions, nullifying the freeze safeguard.

Recommendation: Ensure the freeze state is respected in all share-transfer pathways. Before calling `_transfer` in `swapToSusd1` and `swapToUsd1Plus`, add explicit checks against the vault's frozen-balance mapping and revert if the sender's shares are frozen. Alternatively, refactor these swap functions to utilize the public `transfer`/`transferFrom` methods so that existing freeze enforcement logic is automatically applied.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified. The function `swapToUsd1Plus` now check usable shares before executing the swap.

3.1.4 Asset mismatch between deposit and `sendUnderlying`

Severity: High Risk

Context: SimpleVault.sol#L66-L77, Vault.sol#L165-L173

Description: In the `SimpleVault.sol` contract's `onDepositUnderlying` function, the vault pulls tokens using the user-supplied `underlyingToken` parameter:

```
SafeERC20.safeTransferFrom(IERC20(underlyingToken), from, address(this), underlyingAmount);
```

However, the downstream `sendUnderlying` function in the same `Vault.sol` contract unconditionally transfers the vault's configured `underlying` token:

```
SafeERC20.safeTransfer(IERC20(underlying), portfolio, amount);
```

If `underlyingToken` differs from `underlying`, the vault will either revert (due to insufficient balance of `underlying`) or mistakenly forward tokens it doesn't hold, potentially draining unrelated assets and rendering the vault insolvent for legitimate withdrawals.

Recommendation: Ensure consistency between the deposited and forwarded tokens. Either enforce `underlyingToken == underlying` in `onDepositUnderlying`, or modify `sendUnderlying` to use the same `underlyingToken` received. This alignment prevents accidental reverts and guards against unintended asset drainage.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified. The function `onDepositUnderlying` now transfers the indicated token amount and avoids calling `sendUnderlying` that strictly used for withdrawals.

3.2 Medium Risk

3.2.1 Portfolio weight sum may exceed 100%

Severity: Medium Risk

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

Description: The `updatePortfolios` function only accounts for the weights of newly added portfolios, without clearing or considering any existing portfolio weights.

```
function updatePortfolios(address[] memory  portfolios_, uint256[] memory weights_) public onlyManager {
    uint256 totalWeight;
    uint256 len = portfolios_.length;
    for (uint256 i = 0; i < len; ) {
        bool success = _portfolios.add(portfolios_[i]);
        require(success, "Duplicated portfolio");

        portfolioWeights[portfolios_[i]] = weights_[i];
        totalWeight += weights_[i];

        unchecked { i++; }
    }
    require(totalWeight == Precision, "Invalid total weight");
    emit UpdatePortfolios(portfolios_, weights_);
}
```

If the `_portfolios` set is not empty prior to calling this function, the combined weights (new and existing) may exceed the defined `PRECISION`. This can lead to incorrect total portfolio allocation and violate the intended constraint that the sum of weights must equal `PRECISION`.

Recommendation: Before updating with new portfolio addresses and weights, explicitly clear the previous portfolio data.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified. The `updatePortfolios()` function now includes logic to delete old data.

3.2.2 Swap functions bypass pause and non-transferable state in `sUSD1PlusVault`

Severity: Medium Risk

Context: sUSD1PlusVault.sol#L193-L213, USD1PlusVault.sol#L171-L178

Description: The `sUSD1PlusVault` contract is designed to respect a global `paused` state and a `transferable` flag inherited from `Vault.sol`, which gate any share movements through the public `transfer` and `transferFrom` methods. However, its `swapToSusd1` and `swapToUsd1Plus` functions internally call the `_transfer` method directly, skipping over these checks. As a result, even when the vault is paused or `transferable` is set to `false`, users can still swap and move their shares via these helper functions, effectively nullifying the maintenance and upgrade safeguards.

Recommendation: Modify the swap functions to honor the vault's pause and transferability controls. Either invoke the public `transfer`/`transferFrom` methods (so existing `paused` and `transferable` checks apply) or insert explicit `require(!paused && transferable, "Transfers disabled")` guards before calling `_transfer`. This ensures that no shares can be moved when the vault is paused or transfers are globally disabled.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified.

3.2.3 Native-token vault deposits revert due to unchecked `decimals()` calls

Severity: Medium Risk

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

Description: In the `Vault.sol` contract's deposit logic, the code unconditionally calls `decimals()` on both `underlyingToken` and `underlying`:

```
uint256 decimals = IERC20Metadata(underlyingToken).decimals();
uint256 targetDecimals = IERC20Metadata(underlying).decimals();
```

When `underlying` is configured as the native-token sentinel (e.g. `0xEeee...EEeE`) and `underlyingToken` is any ERC-20 (such as USDC), the first call (`underlyingToken.decimals()`) succeeds, but the second call (`underlying.decimals()`) reverts because the native-token address does not implement the ERC-20 interface. This means any deposit into a vault with a native underlying asset will always revert before accepting funds.

Recommendation: Before calling `decimals()`, detect the native-token sentinel and substitute a hard-coded decimal value (typically 18). For example:

```
uint256 targetDecimals = underlying == NATIVE_TOKEN
    ? 18
    : IERC20Metadata(underlying).decimals();
```

Apply the same pattern for `underlyingToken` if it may ever represent the native asset. This change ensures deposits into native-token vaults proceed without reverting.

Lorenzo: Acknowledged. This will not happen when we actually deploy the product. In actual product deployment, there may be multiple stablecoins as underlying and underlyingToken, and the combination of NATIVE TOKEN and stablecoin will not be used.

Cantina Managed: Acknowledged by Lorenzo team.

3.2.4 Inconsistent use of `underlying` vs. `underlyingToken` in deposit logic

Severity: Medium Risk

Context: SimpleVault.sol#L53-L57

Description: In `onDepositUnderlying` of `SimpleVault.sol`, the branch condition checks the vault's configured `underlying` against `NATIVE_TOKEN`, while the ERC-20 transfer pulls from the user-supplied `underlyingToken` parameter:

```
if (underlying == NATIVE_TOKEN) {
    require(msg.value == underlyingAmount, "send value != deposit amount");
} else {
    SafeERC20.safeTransferFrom(IERC20(underlyingToken), from, address(this), underlyingAmount);
}
```

This mismatch means that when the vault's primary `underlying` is native, the user is forced to send ETH even if they intended to deposit a different token. Conversely, if the vault supports multiple underlying tokens, a deposit of native currency could slip into the ERC-20 path (and vice versa), leading to failed calls or misrouted funds.

Recommendation: Route deposit logic by the actual token being deposited. First validate that the `underlyingToken` parameter matches one of the vault's supported assets, then branch on `underlyingToken == NATIVE_TOKEN` (not on `underlying`). For single-asset vaults, enforce `require(underlyingToken == underlying)`. This ensures the correct transfer method (`msg.value` vs. `safeTransferFrom`) aligns with the user's deposit token.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Ineffective Handling of FoT or Rebasing Tokens

Severity: Low Risk

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

Description: Certain ERC20 tokens may change user's balances over time (positively or negatively) or charge a fee when a transfer is called (FoT tokens). The accounting of these tokens is not handled by Vault.sol and may result in tokens being stuck in Vault or overstating the balance of a user.

Thus, for FoT tokens if all users tried to claim from the Vault there would be insufficient funds and the last user could not withdraw their tokens.

Recommendation: It is recommend documenting clearly that rebasing token should not be used in the protocol.

Alternatively, if it is a requirement to handle rebasing tokens balance checks should be done before and after the transfer to ensure accurate accounting.

Lorenzo: Acknowledged. FOT will not be used.

Cantina Managed: Acknowledged.

3.3.2 Missing Zero Address Check

Severity: Low Risk

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

Description: When assigning a new value to the signer address, there is no validation to prevent it from being set to the zero address. If signer is accidentally or maliciously set to address(0), all signature verifications will effectively be bypassed. This allows any user to confirm or refund deposits on behalf of others without proper authorization, leading to serious security risks.

```
function setSigner(address[] memory signers_, bool[] memory isSigners_) public onlyManager {
    require(signers_.length == isSigners_.length, "mismatch length of signers");

    for (uint256 i = 0; i < signers_.length; i++) {
        signers[signers_[i]] = isSigners_[i];
    }
}
```

Recommendation: Add a check to ensure that the signer address is not set to the zero address during assignment.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified. Zero address checking logic has been added.

3.3.3 Manager Can Freeze Assets Beyond User's Usable Balance

Severity: Low Risk

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

Description: In `freezeShares()`, `balanceOf(account)` was mistakenly used as the maximum freezeable amount, ignoring shares that are already frozen or pending withdrawal. This discrepancy can lead to unexpected freezes and accounting errors.

Recommendation: Consider using `getUsableShares()` instead of `balanceOf()`.

```
uint256 leftShares = getUsableShares(account)
```

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified. The freezeShares() function now correctly uses getUsableShares() to determine the funds a user can freeze.

3.3.4 Dust loss in `alignDepositAmount` due to integer division

Severity: Low Risk

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

Description: In the `alignDepositAmount` function from the `Vault.sol` contract, when `decimals > targetDecimals`, the raw deposit amount is down-scaled using integer division `rawAmount / 10**(decimals - targetDecimals)`. Any remainder from this division, the "dust", is discarded and never credited to the depositor. As a result, users permanently lose these residual amounts when minting LP shares, which may cause the vault's NAV per share to drift over time and potentially benefits subsequent depositors at the expense of earlier ones.

Recommendation: Avoid truncation when scaling amounts. Use a rounding strategy (for example, round-to-nearest) instead of pure integer division, or capture and store any remainder in a per-user dust balance that can be claimed later, ensuring no depositor funds are irreversibly lost.

Lorenzo: Acknowledged.

Cantina Managed: Acknowledged by Lorenzo team.

3.3.5 Locked native currency due to missing `msg.value` guard in ERC-20 deposits

Severity: Low Risk

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

Description: In the `sUSD1PlusVault.sol` contract's inherited `deposit` function (which remains marked `payable`), callers can include native currency even when depositing ERC-20 tokens. The function then executes:

```
SafeERC20.safeTransferFrom(assetToken, from, ceffWallet, underlyingAmount);
```

Since there is no `require(msg.value == 0)` check, any native currency sent alongside a valid ERC-20 deposit remains in the contract's balance after the call succeeds, accumulating over time with no withdrawal path. If the user instead passes the native-token sentinel as `underlyingToken`, the `safeTransferFrom` call reverts.

Recommendation: Prevent native-currency deposits by adding an explicit guard at the start of the function (e.g., `require(msg.value == 0, "No native currency accepted");`) when the vault is configured for ERC-20 assets. Also validate that `underlyingToken` cannot be the native-token sentinel. This change ensures any native currency included in an ERC-20 deposit is immediately rejected, avoiding locked balances.

Lorenzo: Acknowledged.

Cantina Managed: Acknowledged by Lorenzo team.

3.4 Gas Optimization

3.4.1 Missing explicit branch for matching decimal precision in `alignDepositAmount`

Severity: Gas Optimization

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

Description: In the `alignDepositAmount` function of the `Vault.sol` contract, there is a control flow statement handling the case when `decimals > targetDecimals`, but no explicit branch for when `decimals == targetDecimals`. Consequently, the logic falls through to the generic `else` path and executes a no-op multiplication or division by `10**0`, incurring unnecessary gas costs on every invocation.

Recommendation: Add an explicit branch in `alignDepositAmount` that immediately returns the original amount when `decimals == targetDecimals`, and only perform scaling calculations in the < and > scenarios. This change ensures that matching decimal cases consume zero additional gas.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified.

3.4.2 Use custom errors instead of string-based revert messages

Severity: Gas Optimization

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

Description: Across the vault system (e.g., in `Vault.sol`, `CeDeFiManager.sol`, and `SimpleVault.sol`), many `require` and `revert` calls rely on string literals for error reporting. For example:

```
require(msg.sender == owner, "Not owner");
require(!paused, "Pausable: paused");
```

String-based errors inflate bytecode size and increase gas costs on failure.

Recommendation: Use generic custom errors instead of string messages (e.g., `error NotOwner();`) and replace string-based checks with `if (...) revert NotOwner();` to reduce bytecode size and gas usage.

Lorenzo: Acknowledged. This will not happen when we actually deploy the product. In actual product deployment, there may be multiple stablecoins as underlying and underlyingToken, and the combination of NATIVE TOKEN and stablecoin will not be used.

Cantina Managed: Acknowledged by Lorenzo team.

3.5 Informational

3.5.1 Hardhat console import present in code

Severity: Informational

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

Description: In `contracts/CeDeFiManager.sol` at line 12, the development-only debugging import `import "hardhat/console.sol";` is still present. This import and any associated `console.log` calls are intended

for local testing and should not be included in a production release. Retaining them in deployed bytecode increases contract size, raises gas costs, and could inadvertently expose internal state if left behind.

Recommendation: Before releasing to production, remove the `hardhat/console.sol` import and all `console.log` invocations. For any necessary on-chain observability, use properly scoped events instead and keep debugging imports confined to development environments.

Lorenzo: Fixed on commit 2ebe3449.

Cantina Managed: Fix verified.

3.5.2 Unimplemented code

Severity: Informational

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

Description: The `createVault` function in `CeDeFiManager.sol` contains empty branches for `YieldType.PrimeWallet`, `YieldType.DefiProtocol`, and `YieldType.FromFund`, meaning no vault is actually created when those types are selected. Additionally, the `LinkVault` and `CompositVault` contracts are declared but have no implementation, making them effectively unusable.

Recommendation: Treat these branches and abstract contracts as work-in-progress. Either fully implement the vault creation logic for each `YieldType` and flesh out `LinkVault`/`CompositVault`, or add explicit `revert` statements to signal that these options are not yet supported.

Lorenzo: Acknowledged. This will not happen when we actually deploy the product. In actual product deployment, there may be multiple stablecoins as underlying and underlyingToken, and the combination of NATIVE TOKEN and stablecoin will not be used.

Cantina Managed: Acknowledged by Lorenzo team.

3.5.3 Centralize `transferable` check into a modifier

Severity: Informational

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

Description: In the `Vault.sol` contract, the `require(transferable, "Not transferable");` guard appears inline in functions (e.g., at line 203) to enforce the `transferable` flag. Repeating this check across multiple methods adds boilerplate and makes the code harder to maintain.

Recommendation: Define a `whenTransferable` modifier that encapsulates `require(transferable, "Not transferable");` and apply it to each function needing this guard. This removes duplication, improves readability, and ensures a single point of maintenance for the transferability logic.

Lorenzo: Acknowledged. This will not happen when we actually deploy the product. In actual product deployment, there may be multiple stablecoins as underlying and underlyingToken, and the combination of NATIVE TOKEN and stablecoin will not be used.

Cantina Managed: Acknowledged by Lorenzo team.

3.5.4 Add pausable state to `sUSD1PlusVault` and `USD1PlusVault`

Severity: Informational

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

Description: The `sUSD1PlusVault.sol` contract's key functions `confirmShare`, `refundShare`, `swapToSusd1`, and `swapToUsd1Plus` are not protected by the vault's pause mechanism, allowing share settlements, refunds, or swaps even when the vault is paused. Similarly, the `USD1PlusVault.sol` contract relies on these operations but does not itself implement any pausable guard. This gap undermines the intended emergency-stop and maintenance capabilities provided by the base `Vault` contract's pause functionality.

Recommendation: Apply the existing pause control (`whenNotPaused` modifier or `require(!paused)`) to all of the above functions in both `sUSD1PlusVault.sol` and `USD1PlusVault.sol`. Ensuring these entry points honor the paused state prevents any share or asset movements during maintenance or in response to emergencies.

Lorenzo: Acknowledged. This will not happen when we actually deploy the product. In actual product deployment, there may be multiple stablecoins as underlying and underlyingToken, and the combination of NATIVE TOKEN and stablecoin will not be used.

Cantina Managed: Acknowledged by Lorenzo team.

3.5.5 Emit event in `setSigner` function

Severity: Informational

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

Description: In `CeDeFiManager.sol`, the `setSigner` function updates the contract's authorized signer without emitting any event to signal this change. As a result, off-chain services and auditors cannot detect when the signer is rotated, reducing transparency and making it harder to track critical governance updates.

Recommendation: Introduce a dedicated event (e.g. `SignerUpdated`) that captures both the previous and new signer addresses, and emit this event at the end of the `setSigner` function. This will ensure every signer rotation is logged on-chain and visible to off-chain monitors.

Lorenzo: Acknowledged. This will not happen when we actually deploy the product. In actual product deployment, there may be multiple stablecoins as underlying and underlyingToken, and the combination of NATIVE TOKEN and stablecoin will not be used.

Cantina Managed: Acknowledged by Lorenzo team.