CERTIK

Lorenzo Protocol - Audit

Security Assessment

CertiK Assessed on Aug 8th, 2025

CertiK Assessed on Aug 8th, 2025

Lorenzo Protocol - Audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Vault	EVM Compatible	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 08/08/2025	N/A

CODEBASE

0fd0b768537e52951bdf81aa71d33132e9615eac

View All in Codebase Page

Vulnerability Summary

18	10	1	7	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

3	Centralization	3 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
3	Medium	3 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
6	Minor	4 Resolved, 1 Partially Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
6	Informational	3 Resolved, 3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | LORENZO PROTOCOL - AUDIT

CODEBASE | LORENZO PROTOCOL - AUDIT

Repository

0fd0b768537e52951bdf81aa71d33132e9615eac

AUDIT SCOPE | LORENZO PROTOCOL - AUDIT

9 files audited ● 5 files with Acknowledged findings ● 4 files without findings

ID	Repo	File	SHA256 Checksum
USD	Lorenzo-Protocol/OTF-Contract	contracts/vaults/USD1/USD1PlusVault.sol	52a17bd3d81eb4ddcd495d78f8e95e0c95a18e65bfbf4f8e5b05c1948e8a9aaf
USP	Lorenzo-Protocol/OTF-Contract	contracts/vaults/USD1/sUSD1PlusVault.sol	1d084607065b02d96922b240444d0314eced534cb03b12e53c4192e9538be70d
SVO	Lorenzo-Protocol/OTF-Contract	contracts/vaults/SimpleVault.sol	dd2c6031869628196f7d3ee79a3312acbae26dbc7b79f2a22471b2019ed112e4
VOT	Lorenzo-Protocol/OTF-Contract	contracts/vaults/Vault.sol	86dae8436809ada23fece1a7092ab528543c4bf7c5e574252fe1249cd6c87662
CDF	Lorenzo-Protocol/OTF-Contract	contracts/CeDeFiManager.sol	2e5d317f7ea6b45518bac267e8523fbc537bdf6cefce699b4dcef50dd9f4e83e
VBO	Lorenzo-Protocol/OTF-Contract	contracts/vaults/VaultBase.sol	ff202ffd97359159384ec5ec3ca7d179fd320ca89e94517bfdb4f93a84c1fb17
CDM	Lorenzo-Protocol/OTF-Contract	contracts/CeDeFiManagerBase.sol	56704e97eeec959dd4d07c8dc21f4707d1dd0a4f714a24871e129a43002d1425
CVO	Lorenzo-Protocol/OTF-Contract	contracts/funds/CompositVault.sol	f0e8349fd502cea59ce76fbecf646426845813068ef9f8f045dc751b481f1b81
LVO	Lorenzo-Protocol/OTF-Contract	contracts/funds/LinkVault.sol	70fa88964f99897a9972a207f1fa55e2161d5cb8796bc69bc704fe9acf7ffdb6

APPROACH & METHODS | LORENZO PROTOCOL - AUDIT

This report has been prepared for Lorenzo to discover issues and vulnerabilities in the source code of the Lorenzo Protocol - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | LORENZO PROTOCOL - AUDIT

Overview

The **Lorenzo Protocol** implements a custody-based rebasing vault system that issues a stable token (`USD1+`) backed by deposits in a secure off-chain-managed vault (`sUSD1`). The protocol integrates on-chain token logic with off-chain compliance via KYT (Know Your Transaction) verification. It supports deposit, minting, transfer, and redemption of shares using NAV-based accounting. The system is modular, upgradeable, and designed to be KYT-compliant and custodially backed. The core components are described below:

- **USD1PlusVault**
  A rebasing ERC20 token contract that represents ownership of NAV-based shares in the `sUSD1PlusVault` . It abstracts share valuation to a stable 1:1 representation for users. It includes mechanisms for minting, redeeming, freezing user balances, and interfacing with the `sUSD1PlusVault` for swapping shares.

- **sUSD1PlusVault**
  A custody-focused vault where users deposit USD1 tokens to receive pending shares, which require signature-based KYT confirmation to be minted. It supports confirming or refunding shares based on signed messages, and manages internal accounting of share states. It also handles two-way swaps between `USD1+` and `sUSD1` tokens.

- **SimpleVault**
  The base vault contract for managing yield-bearing assets or custodial portfolios. It maintains a portfolio with a single strategy, tracks NAV per settlement period, and implements share minting, burning, and valuation logic. It is inherited by `sUSD1PlusVault` and interacts with underlying portfolios.

- **CeDeFiManager**
  A centralized management controller responsible for configuring portfolios, managing permissions, initiating upgrades, and overseeing NAV settlement logic across the vault ecosystem. It is the central role-based admin entity within the protocol.

External Dependencies

The project relies on a few external contracts or addresses to fulfill the needs of its business logic.

- @openzeppelin/contracts-upgradeable
- @openzeppelin/contracts

It is assumed that these contracts and addresses are trusted, implemented, and properly configured throughout the entire project.

Privileged Functions

In the **Lorenzo Protocol**, the admin roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the findings `Centralization Related Risks` .

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan.

Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project. To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | LORENZO PROTOCOL - AUDIT

18	0	3	0	3	6	6
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Lorenzo Protocol - Audit. Through this audit, we have uncovered 18 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
LPA-12	**Centralization Risks**	**Centralization**	**Centralization**	**Acknowledged**
LPA-13	**Irreversible Centralized Control Over `unitNav` Possibly Disrupt Vault Operations**	**Centralization, Logical Issue**	**Centralization**	**Acknowledged**
LPA-20	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	**Acknowledged**
LPA-02	Users Can Bypass Blacklist Via `swapToUsd1Plus` In `sUSD1PlusVault`	Logical Issue	Medium	Resolved
LPA-04	Share Inflation Attack In `sUSD1PlusVault`	Logical Issue	Medium	Resolved
LPA-14	Redeem Ignores Frozen-Share Restrictions Allowing Withdrawal Of Frozen Shares	Logical Issue	Medium	Resolved
LPA-03	Potential Front-Running Of `setUnitNav` Enables Withdraw Reward Manipulation	Logical Issue	Minor	Acknowledged
LPA-05	Inherited Contracts Not Initialized In Initializer	Logical Issue	Minor	Resolved
LPA-06	Missing Validation In `getUnitNav` Allows Unsafe Access To Uninitialized NAV Data	Coding Issue	Minor	Resolved

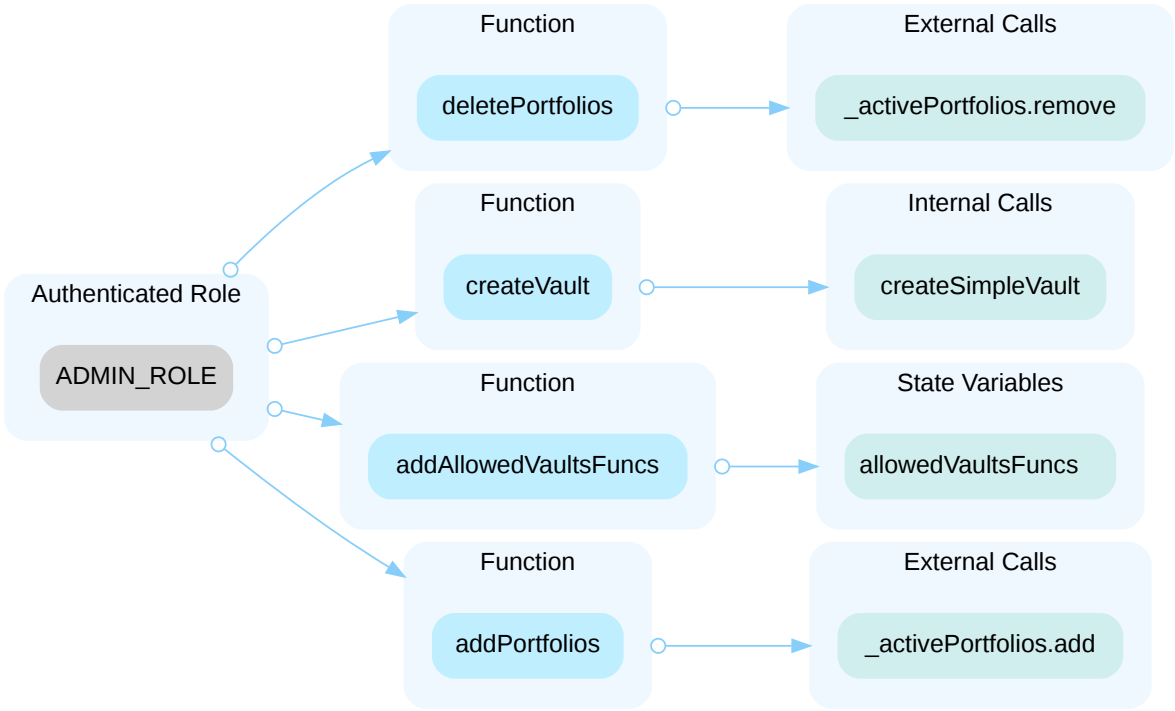
ID	Title	Category	Severity	Status
LPA-07	Unhandled ETH In `onDepositUnderlying` Function In `sUSD1PlusVault` And `SimpleVault` Contracts	Logical Issue	Minor	Partially Resolved
LPA-15	`freezeShares()` Uses `balanceOf()` Instead Of `getUsableShares()` , Potentially Over-Freezing Inaccessible Shares	Logical Issue	Minor	Resolved
LPA-17	Missing Zero Address Validation	Volatile Code	Minor	Resolved
LPA-08	Self-Administered Role Allows For Complete Role Takeover	Design Issue	Informational	Acknowledged
LPA-09	Unnecessary `view` Modifier In `_authorizeUpgrade` Function	Coding Style	Informational	Resolved
LPA-10	Potential Hash Collision In `createSimpleVault` 'S Salt Calculation	Coding Issue	Informational	Resolved
LPA-11	Inconsistent Portfolio Accounting Between Deposit And Withdrawal	Inconsistency	Informational	Acknowledged
LPA-16	Missing Length Validation In `updatePortfolios()` May Cause Out-Of-Bounds Access	Volatile Code	Informational	Resolved
LPA-19	Missing Interface Implementatione	Coding Issue	Informational	Acknowledged

LPA-12 | CENTRALIZATION RISKS

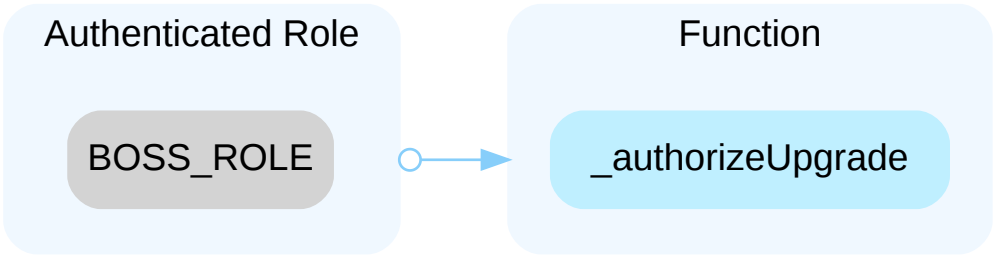
Category	Severity	Location	Status
Centralization	Centralization	contracts/CeDeFiManager.sol (init): 37, 41, 52, 63, 74, 130; contracts/vaults/USD1/USD1PlusVault.sol (init): 66, 130, 142, 153; contracts/vaults/Vault.sol (init): 44, 67, 82, 90, 98, 109, 120, 131, 142, 147, 163	Acknowledged

Description

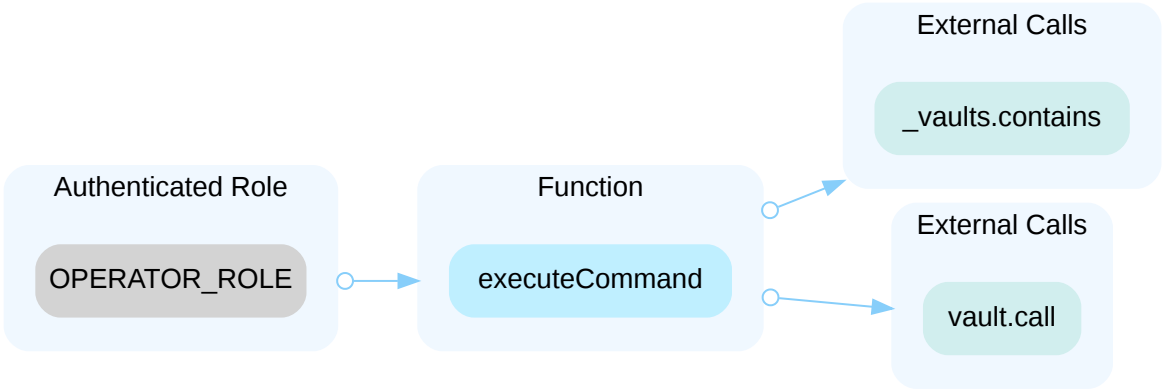
In the contract `CeDeFiManager`, the role `ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN_ROLE` account may allow the hacker to take advantage of this authority and delete specified portfolios, create a new vault, add allowed vault functions, and add portfolios to the active list.



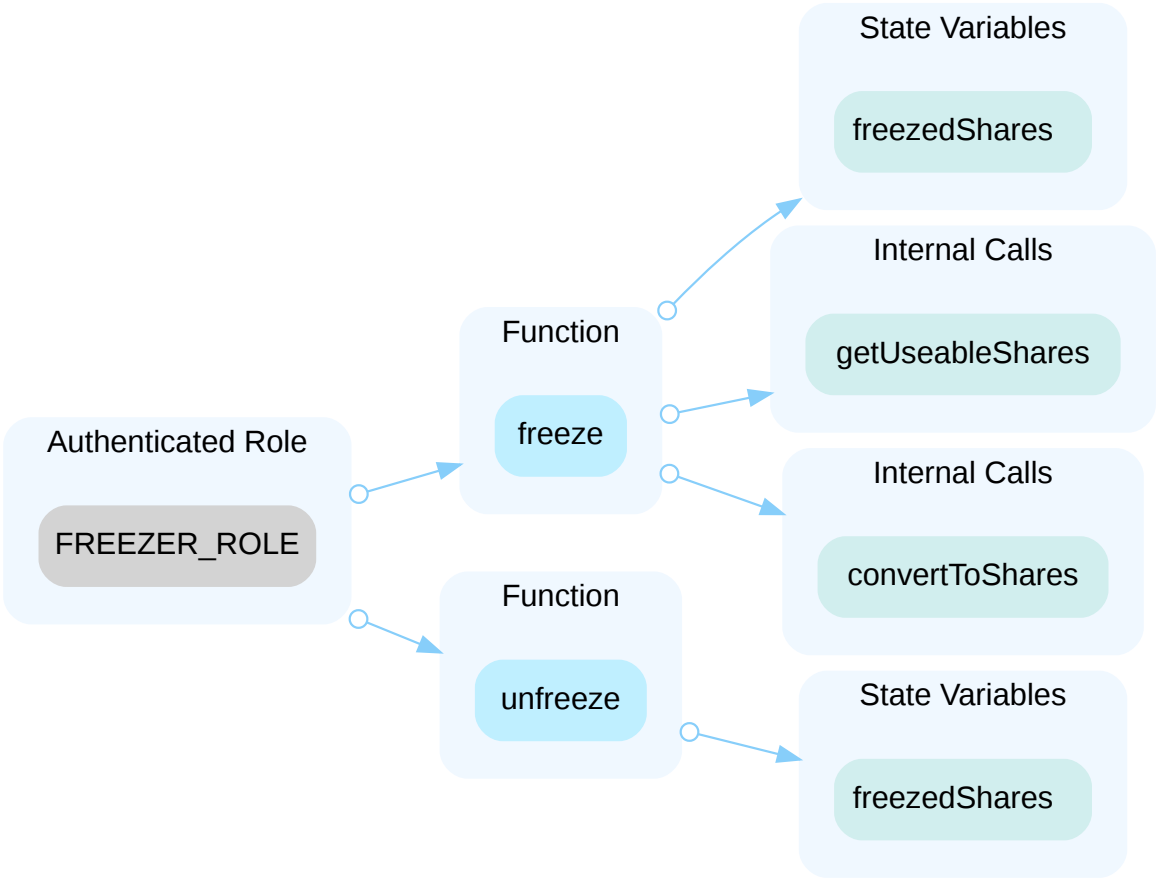
In the contract `CeDeFiManager`, the role `BOSS_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `BOSS_ROLE` account may allow the hacker to take advantage of this authority and authorize contract upgrades for the boss role.



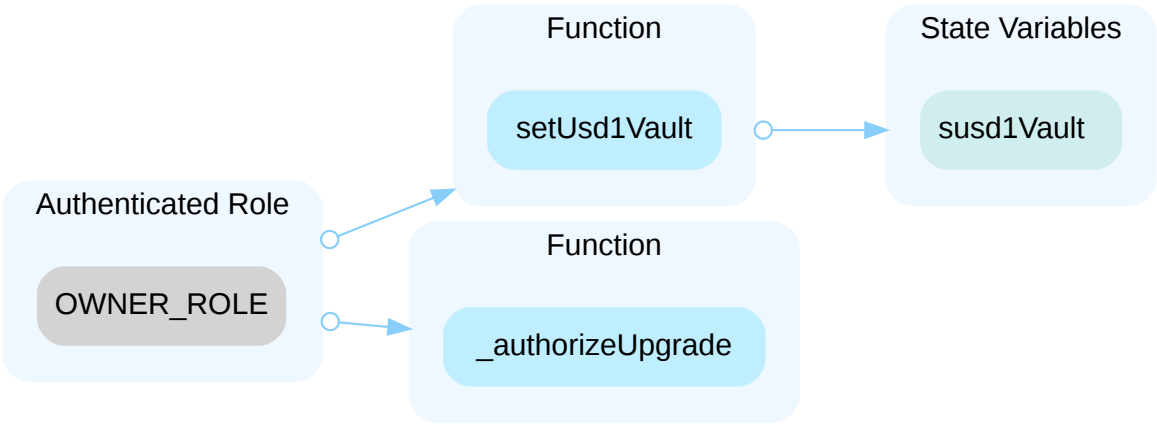
In the contract `CeDeFiManager`, the role `OPERATOR_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `OPERATOR_ROLE` account may allow the hacker to take advantage of this authority and execute commands on specified vaults.



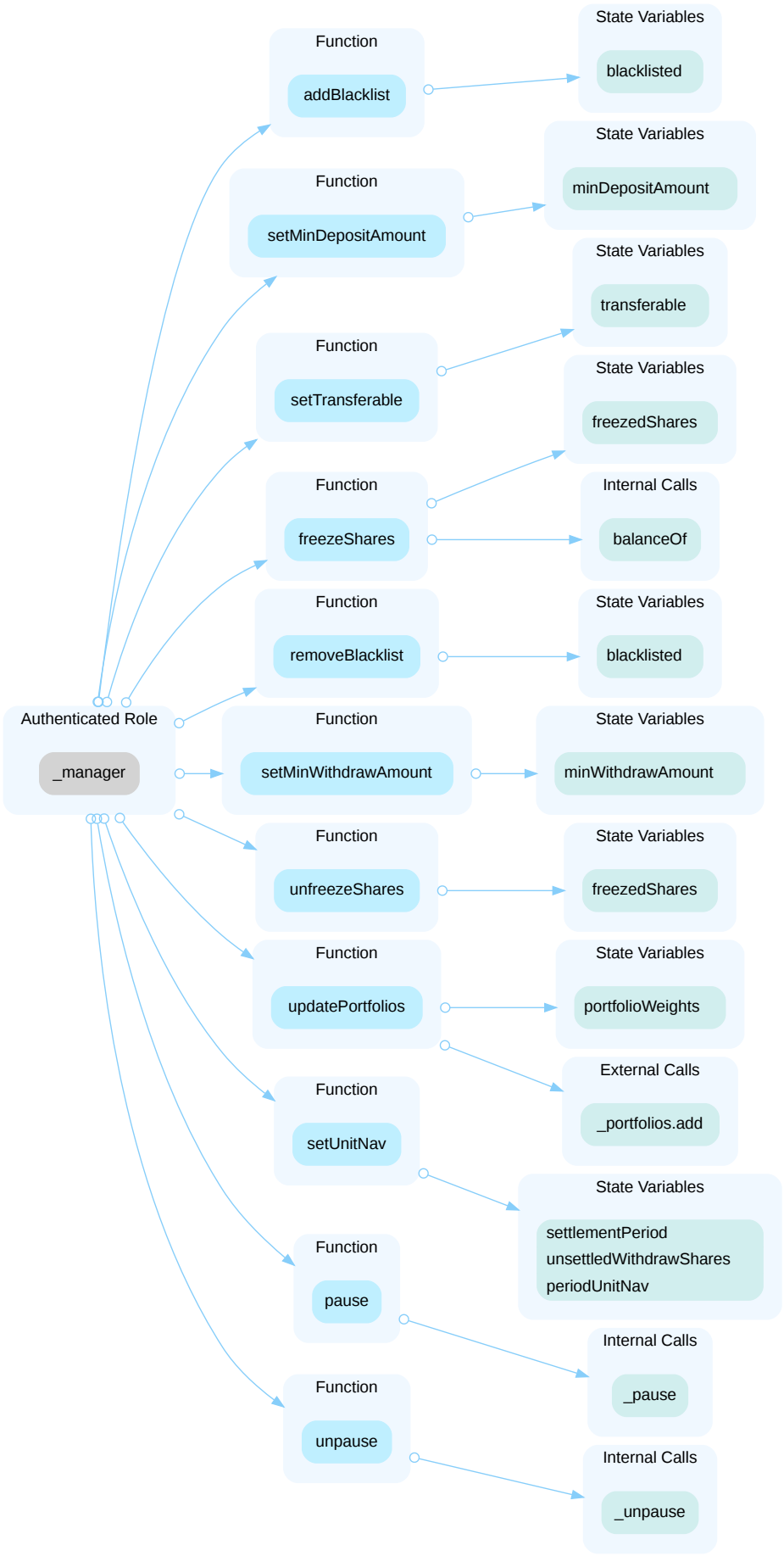
In the contract `USD1PlusVault`, the role `FREEZER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `FREEZER_ROLE` account may allow the hacker to take advantage of this authority and freeze or unfreeze specified amounts of shares for an account.



In the contract `USD1PlusVault`, the role `OWNER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `OWNER_ROLE` account may allow the hacker to take advantage of this authority and set the USD1 vault address, authorize contract upgrade to a new implementation.



In the contract `Vault`, the role `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_manager` account may allow the hacker to take advantage of this authority and add users to the blacklist, set the minimum deposit amount, set the transferable state, freeze shares for a given account, remove users from the blacklist, set the minimum withdrawal amount, unfreeze specified shares for an account, update portfolios and set their weights, set unit NAV if not settled already, pause the contract, and unpause the contract.


Function
addBlacklist
State Variables
blacklisted
Function
setMinDepositAmount
State Variables
minDepositAmount
Function
setTransferable
State Variables
transferable
Function
freezeShares
State Variables
freezedShares
Internal Calls
balanceOf
Function
removeBlacklist
State Variables
blacklisted
Authenticated Role
_manager
Function
setMinWithdrawAmount
State Variables
minWithdrawAmount
Function
unfreezeShares
State Variables
freezedShares
Function
updatePortfolios
State Variables
portfolioWeights
External Calls
_portfolios.add
Function
setUnitNav
State Variables
settlementPeriod
unsettledWithdrawShares
periodUnitNav
Function
pause
Internal Calls
_pause
Function
unpause
Internal Calls
_unpause

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Lorenzo, 07/15/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> Issue acknowledged. I won't make any changes for the current version.

LPA-13 IRREVERSIBLE CENTRALIZED CONTROL OVER `unitNav` POSSIBLY DISRUPT VAULT OPERATIONS

Category	Severity	Location	Status
Centralization, Logical Issue	**Centralization**	**contracts/vaults/Vault.sol (init): 73**	**Acknowledged**

Description

The `setUnitNav()` function allows the manager to set the unit net asset value (unitNav) for each settlement period. While the function ensures that the value is non-zero and only set once per period, it lacks both upper and lower bound validation, allowing arbitrary values to be submitted.

Once set, the `unitNav` value is permanent and cannot be corrected, creating a centralized, irreversible single point of failure. If a mistake is made, the consequences affect all depositors and withdrawers for that settlement period.

If `unitNav` is set too high, user deposits will result in:

```
    function mintShares(address to, uint256 underlyingAmount) internal
returns(uint256 sharesAmount, uint256 unitNav) {
        unitNav = getUnitNav(settlementPeriod);
        require(unitNav > 0, "oops: zero unit nav");
@>      sharesAmount = (underlyingAmount * Precision) / unitNav;

        _mint(to, sharesAmount);
    }
```

which may evaluate to zero, causing users to receive no shares in return for their deposit.

If `unitNav` is set too low, user withdrawals will compute:

```
    function burnShares(address from, uint128 requestId, uint256 sharesAmount)
internal returns(uint256) {
        uint256 unitNav = periodUnitNav[requestId];
@>      uint256 amount = (sharesAmount * unitNav) / Precision;
        // burn shares and return the amount of underlying asset
        _burn(from, sharesAmount);
        return amount;
    }
```

which may round down to zero, burning user shares without issuing any underlying assets. This misconfiguration can lock user funds or disrupt vault accounting irreversibly.

Moreover, the `unitNav` directly determines whether users gain or lose underlying assets from interacting with the vault. Mispricing this value can lead to system-wide financial inconsistencies, unfair user treatment, or complete locking of user

funds.

Recommendation

Because `setUnitNav()` is a single-call, irreversible operation, any error — accidental or malicious — leads to permanent vault misbehavior. This centralization risk undermines the reliability and integrity of the vault system and should be carefully mitigated.

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Lorenzo, 07/15/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> Yes, the owner would be a multi sig wallet. and I will add a new method to reset NAV but not update period.

[Lorenzo, 07/24/2025]: Add a new function `resetUnitNav` which just reset the unit nav but not change period in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-20 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	Centralization	contracts/CeDeFiManager.sol (init): 14; contracts/vaults/SimpleVault.sol (init): 11; contracts/vaults/USD1/USD1PlusVault.sol (init): 15; contracts/vaults/USD1/sUSD1PlusVault.sol (init): 15	Acknowledged

Description

In the contracts `CeDeFiManager`, `SimpleVault`, `USD1PlusVault`, and `sUSD1PlusVault`, the role `admin` has the authority to update the implementation contracts.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority and change the implementation contract, which is pointed by proxy, and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Lorenzo, 07/15/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> all the `owner` and `admin` will be set a gnosis address.

LPA-02 USERS CAN BYPASS BLACKLIST VIA `swapToUsd1Plus` IN `sUSD1PlusVault`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/vaults/USD1/sUSD1PlusVault.sol (init): 199~210	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/sUSD1PlusVault.sol`
- `contracts/vaults/USD1/USD1PlusVault.sol`
- `contracts/vaults/Vault.sol`

The `sUSD1PlusVault` contract inherits blacklist restrictions from the `Vault` contract, which enforces checks in `transfer()` and `transferFrom()` functions. However, a critical vulnerability exists where users can bypass these blacklist restrictions by calling `swapToUsd1Plus()`, which internally uses `_transfer()` without any blacklist validation. This allows blacklisted users to move their funds out of the system by converting sUSD1 to USD1+ tokens.

The Vault contract implements blacklist checks in:

1. `transfer()` → Checks `msg.sender` against the blacklist.

`contracts/vaults/Vault.sol`

```
182 function transfer(address recipient, uint256 amount)
183         public
184         virtual
185         override(IERC20, ERC20Upgradeable)
186         whenNotPaused
187         notBlacklisted
188         returns (bool)
189     {
190         require(transferable, "Not transferable");
191         require(getUsableShares(msg.sender) >= amount, "Insufficient balance");
192         return super.transfer(recipient, amount);
193     }
```

2. `transferFrom()` → Checks both `from` and `msg.sender` against the blacklist.

`contracts/vaults/Vault.sol`

```
197 function transferFrom(address from, address to, uint256 value)
198         public
199         virtual
200         override(IERC20, ERC20Upgradeable)
201         whenNotPaused
202         notBlacklisted
203         returns (bool)
204     {
205         require(transferable, "Not transferable");
206         require(!blacklisted[from], "from is blacklisted");
207         require(getUsableShares(from) >= value, "Insufficient balance");
208         return super.transferFrom(from, to, value);
209     }
```

These functions use the `notBlacklisted` modifier, ensuring blocked users cannot transfer shares.

The `sUSD1PlusVault` contract inherits blacklist restrictions from the `Vault` contract. However, the `swapToUsd1Plus()` function in `sUSD1PlusVault` contract allows users to convert sUSD1 to USD1+ by transferring sUSD1 shares to `usd1PlusVault` via `_transfer()` and minting equivalent USD1+ tokens to the receiver:

`contracts/vaults/USD1/sUSD1PlusVault.sol`

```
199 function swapToUsd1Plus(address receiver, uint256 sharesAmount) public {
200         require(receiver != address(0), "invalid receiver");
201         require(sharesAmount > 0, "invalid shares amount");
202 
203         // lock the shares
204         _transfer(msg.sender, address(usd1PlusVault), sharesAmount);
205 
206         // mint the shares to the receiver
207         usd1PlusVault.mint(receiver, sharesAmount);
208 
209         emit SwapSusd1ToUsd1Plus(msg.sender, receiver, sharesAmount);
210     }
```

The `_transfer()` is an internal function that does not enforce blacklist checks. After minting equivalent USD1+ tokens to the user, the user can call `redeem` function in `USD1PlusVault` contract to convert USD1+ tokens back to sUSD1 to any `to` address:

`contracts/vaults/USD1/USD1PlusVault.sol`

```
171 function redeem(address to, uint256 onePlusAmount) public {
172         uint256 sharesAmount = convertToShares(onePlusAmount);
173         _burn(msg.sender, sharesAmount);
174 
175         susd1Vault.swapToSusd1(to, sharesAmount);
176 
177         emit RedeemUsd1Plus(to, onePlusAmount);
178     }
```

So a blacklisted user can first call `swapToUsd1Plus()` function in `sUSD1PlusVault` and then call `redeem` function in `USD1PlusVault` to move his shares to any address, bypassing blacklist restrictions.

Recommendation

Consider redesigning the logic of `swapToUsd1Plus` function.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue by adding `notBlacklisted` modifier in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-04 | SHARE INFLATION ATTACK IN `sUSD1PlusVault`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/vaults/Vault.sol (init): 264~299	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/sUSD1PlusVault.sol`
- `contracts/vaults/USD1/USD1PlusVault.sol`
- `contracts/vaults/Vault.sol`

The `requestWithdraw` function in the vault system records withdrawal requests in `periodWithdrawShares[requestId]`. The `swapToUsd1Plus` function transfers shares directly without checking whether these shares are in pending withdraw requests. This allows a malicious user to:

1. Request a withdrawal, increasing `periodWithdrawShares`.
2. Transfer the same shares (e.g., via `swapToUsd1Plus`) to another account.
3. Repeat the withdrawal request, artificially inflating `periodWithdrawShares`.
4. Cause `unsettledWithdrawShares` to grow indefinitely, potentially leading to an underflow in `totalSupply()`.

This flaw enables share inflation, incorrect accounting, and possible denial-of-service (DoS) via `totalSupply()` reverts.

The `requestWithdraw` function in `Vault` records shareAmount in `periodWithdrawShares[requestId]` and does not burn shares, allowing them to remain transferable:

`contracts/vaults/Vault.sol`

```
264  function requestWithdraw(uint256 shareAmount)
265          public
266          virtual
267          whenNotPaused
268          notBlacklisted
269          lock
270          returns (uint128)
271      {
272          require(shareAmount > 0, "share amount is zero");
273          uint256 sharesBalance = getUsableShares(msg.sender);
274          if (shareAmount != sharesBalance) {
// make sure sender can withdraw all shares
275              require(shareAmount >= minWithdrawAmount,
"withdraw amount too small");
276          }
277          require(sharesBalance >= shareAmount, "Insufficient balance");
278
279
// to record the request information, merge multiple requests into one request
280          uint128 requestId = settlementPeriod;
281          _withdrawRequestIds[msg.sender].add(requestId);
282
283          WithdrewRequest storage request = withdrewRequests[msg.sender][
requestId];
284          request.requestShares += shareAmount;
285          request.targetPeriod = requestId;
286          request.requestTimestamp = uint64(block.timestamp);
287
// to record the total withdraw shares of this period, and sub it while this period
is settled

288          // so these shares stop yielding
289          periodWithdrawShares[requestId] += shareAmount;
290
291          emit WithdrawRequested(
292              requestId,
293              msg.sender,
294              shareAmount,
295              block.timestamp
296          );
297
298          return requestId;
299      }
```

The `swapToUsd1Plus` function in `sUSD1PlusVault` transfers shares directly without checking whether these shares are in pending withdraw requests:

`contracts/vaults/USD1/sUSD1PlusVault.sol`

```
199 function swapToUsd1Plus(address receiver, uint256 sharesAmount) public {
200         require(receiver != address(0), "invalid receiver");
201         require(sharesAmount > 0, "invalid shares amount");
202 
203         // lock the shares
204         _transfer(msg.sender, address(usd1PlusVault), sharesAmount);
205 
206         // mint the shares to the receiver
207         usd1PlusVault.mint(receiver, sharesAmount);
208 
209         emit SwapSusd1ToUsd1Plus(msg.sender, receiver, sharesAmount);
210     }
```

The `redeem` function in `USD1PlusVault` can return shares to any account:

`contracts/vaults/USD1/USD1PlusVault.sol`

```
171 function redeem(address to, uint256 onePlusAmount) public {
172         uint256 sharesAmount = convertToShares(onePlusAmount);
173         _burn(msg.sender, sharesAmount);
174 
175         susd1Vault.swapToSusd1(to, sharesAmount);
176 
177         emit RedeemUsd1Plus(to, onePlusAmount);
178     }
```

The `setUnitNav()` function in `Vault` moves `periodWithdrawShares[requestId]` to `unsettledWithdrawShares`:

`contracts/vaults/Vault.sol`

```
 67 function setUnitNav(uint256 unitNav) external onlyManager {
 68         uint128 index = settlementPeriod;
 69         require(periodUnitNav[index] == 0, "period settled already");
 70         // FIXME: should do this check??????
 71         require(unitNav != 0, "invalid unit nav");
 72 
 73         periodUnitNav[index] = unitNav;
 74         unsettledWithdrawShares += periodWithdrawShares[index];
 75 
 76         ++settlementPeriod;
 77 
 78         emit SetUnitNav(index, unitNav);
 79     }
```

The `totalSupply()` in `Vault` subtracts `unsettledWithdrawShares` from the `rawSupply`:

`contracts/vaults/Vault.sol`

```
function totalSupply()
        public
        virtual
        view
        override(IERC20, ERC20Upgradeable)
        returns (uint256)
    {
        uint256 rawSupply = super.totalSupply();
        return rawSupply - unsettledWithdrawShares;
    }

```

As a result, a user can:

1. Call `requestWithdraw(X)` → `periodWithdrawShares += X`.
2. Transfer shares via `swapToUsd1Plus(X)` and `redeem` to a new account.
3. Call `requestWithdraw(X)` again from the new account → `periodWithdrawShares += X (duplicate counting)`.

Repeating this inflates `periodWithdrawShares` and `unsettledWithdrawShares` and `totalSupply()` will underflow if `unsettledWithdrawShares > rawSupply`. Any external contract depends on `totalSupply()` will encounter **denial-of-service (DoS)**.

Recommendation

Consider checking whether these shares are transferable in `swapToUsd1Plus` function.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-14 REDEEM IGNORES FROZEN-SHARE RESTRICTIONS ALLOWING WITHDRAWAL OF FROZEN SHARES

Category	Severity	Location	Status
Logical Issue	Medium	contracts/vaults/USD1/USD1PlusVault.sol (init): 171~178	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/USD1PlusVault.sol`

The `redeem` function in the `USD1PlusVault` contract lacks a critical check to ensure that users cannot redeem frozen shares. This oversight allows users to bypass the freezing mechanism, potentially leading to unauthorized redemptions and undermining the intended security controls.

The `transfer` and `transferFrom` enforce frozen share restrictions:

`contracts/vaults/USD1/USD1PlusVault.sol`

```
104  function transfer(address to, uint256 onePlusAmount) public override(
ERC20Upgradeable) returns (bool) {
105          uint256 shares = convertToShares(onePlusAmount);
106          require(getUseableShares(msg.sender) >= shares,
"can not transfer so much shares");
107          return super.transfer(to,shares);
108      }
109
110
// @dev transferFrom onePlusAmount from the source address to the target address,
internally convert to shares and transfer

111      function transferFrom(address from, address to, uint256 onePlusAmount)
public override(ERC20Upgradeable) returns (bool) {
112          uint256 shares = convertToShares(onePlusAmount);
113          require(getUseableShares(from) >= shares,
"can not transfer so much shares");
114          return super.transferFrom(from, to, shares);
115      }
116
117      function getUseableShares(address account) public view returns (uint256) {
118          return super.balanceOf(account) - freezedShares[account];
119      }
```

The `redeem` function allows users to burn their shares (USD1+) in exchange for sUSD1. However, unlike other functions (`transfer`, `transferFrom`), it does not verify whether the shares being redeemed are frozen:

`contracts/vaults/USD1/USD1PlusVault.sol`

```
171  function redeem(address to, uint256 onePlusAmount) public {
172          uint256 sharesAmount = convertToShares(onePlusAmount);
173          _burn(msg.sender, sharesAmount);
174
175          susd1Vault.swapToSusd1(to, sharesAmount);
176
177          emit RedeemUsd1Plus(to, onePlusAmount);
178      }
```

The function does not call `getUseableShares(msg.sender)` to ensure the shares are not frozen.

Impact

1. Bypassing Freezing Mechanism: Users can redeem shares that should be locked.
2. Security Risk: If shares are frozen for compliance (e.g., suspicious activity), they could still be redeemed, violating intended restrictions.
3. Protocol Integrity: The freeze feature becomes unreliable, affecting trust in the system.

Recommendation

Consider adding a check for frozen shares in `redeem` function.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue by adding a check for frozen shares in `redeem` function in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-03 POTENTIAL FRONT-RUNNING OF `setUnitNav` ENABLES WITHDRAW REWARD MANIPULATION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/vaults/Vault.sol (init): 67, 264	Acknowledged

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/Vault.sol`

The vault contract allows users to deposit underlying assets and mint shares based on a unitNav value defined per `settlementPeriod`. The `unitNav` is set by the manager through the `setUnitNav()` function, and cannot be updated or reverted once committed. This value also determines how many underlying tokens a user receives when withdrawing shares.

```
 339     function mintShares(address to, uint256 underlyingAmount) internal returns(
uint256 sharesAmount, uint256 unitNav) {
 340         unitNav = getUnitNav(settlementPeriod);
 341         require(unitNav > 0, "oops: zero unit nav");
 342 @>      sharesAmount = (underlyingAmount * Precision) / unitNav;
 343
 344         _mint(to, sharesAmount);
 345     }
```

```
 350     function burnShares(address from, uint128 requestId, uint256 sharesAmount)
internal returns(uint256) {
 351         uint256 unitNav = periodUnitNav[requestId];
 352 @>      uint256 amount = (sharesAmount * unitNav) / Precision;
 353         // burn shares and return the amount of underlying asset
 354         _burn(from, sharesAmount);
 355         return amount;
 356     }
```

However, because the `setUnitNav()` call is a publicly observable transaction, a malicious actor could front-run it by calling `requestWithdraw()` immediately before `setUnitNav()` is mined. Since `requestWithdraw()` binds the user's withdrawal

request to the current `settlementPeriod`, which will be settled by the upcoming `setUnitNav()` transaction, the attacker can exploit this by anticipating a high unitNav value.

Scenario

Exploit Path:

1. An attacker observes an incoming `setUnitNav(high_value)` transaction in the mempool, which sets a higher-than-expected unitNav.
2. Before the `setUnitNav()` transaction is mined, the attacker submits a requestWithdraw() transaction. This binds their withdrawal to the current `settlementPeriod`.
3. Once `setUnitNav()` is confirmed on-chain, the `settlementPeriod` is updated with the inflated unitNav. The attacker's withdrawal request is now tied to this higher value.
4. When the attacker later executes `withdraw()`, their shares are burned using the artificially high unitNav, resulting in a payout of more underlying assets than they fairly earned.

Recommendation

Consider automating NAV computation based on on-chain portfolio valuations and price oracles, reducing human error and centralization risk.

Alleviation

[Lorenzo, 07/15/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> yes, we know this risk. but users only can withdraw after period + 1 is settled, and no one knows the unit nav then.

[Lorenzo, 07/24/2025]: as the discussion in TG, the user must wait for two period before finish the withdraw request. so no front-run risk for this function.

LPA-05 | INHERITED CONTRACTS NOT INITIALIZED IN INITIALIZER

Category	Severity	Location	Status
Logical Issue	Minor	contracts/vaults/USD1/USD1PlusVault.sol (init): 16	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/USD1PlusVault.sol`

Contract `USD1PlusVault` extends `UUPSUpgradeable`, but the extended contract is not initialized by the current contract.

`contracts/vaults/USD1/USD1PlusVault.sol`

```
15 contract USD1PlusVault is
16     UUPSUpgradeable,
17     ERC20Upgradeable,
18     AccessControlUpgradeable
19 {
```

`contracts/vaults/USD1/USD1PlusVault.sol`

```
43  function initialize(
44          address _owner,
45          address _susd1Vault,
46          address[] memory _freezers
47      ) public initializer {
48          require(_owner != address(0), "invalid owner");
49          require(_susd1Vault != address(0), "invalid usd1 vault");
50
51          __ERC20_init("Lorenzo USD1+", "USD1+");
52          __AccessControl_init();
53
54          _setRoleAdmin(OWNER_ROLE, OWNER_ROLE);
55          _setRoleAdmin(FREEZER_ROLE, OWNER_ROLE);
56
57          _grantRole(OWNER_ROLE, _owner);
58          _grantRole(FREEZER_ROLE, _owner);
59
60          for (uint256 i; i < _freezers.length; i++) {
61              _grantRole(FREEZER_ROLE, _freezers[i]);
62          }
63          susd1Vault = IsUSD1PlusVault(_susd1Vault);
64      }
```

Generally, the initializer function of a contract should always call all the initializer functions of the contracts that it extends.

Recommendation

We recommend initializing `UUPSUpgradeable`.

Alleviation

[Lorenzo, 07/25/2025]: The team heeded the advice and resolved the issue by initializing `UUPSUpgradeable` in commit fd93c26740dfed3f2731456f6209a960373ca495.

LPA-06 MISSING VALIDATION IN `getUnitNav` ALLOWS UNSAFE ACCESS TO UNINITIALIZED NAV DATA

Category	Severity	Location	Status
Coding Issue	Minor	contracts/vaults/Vault.sol (init): 386	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/Vault.sol`

Accessing `periodUnitNav[period - 1]` without validation in a public function can lead to unintended behavior if `period == 0`.

```
377     function getUnitNav(uint128 period) public view returns (uint256) {
378         if (period == 1) {
379             if (underlying == NATIVE_TOKEN) {
380                 return 10 ** 18;
381             } else {
382                 IERC20Metadata underlyingToken = IERC20Metadata(underlying);
383                 return 10 ** underlyingToken.decimals();
384             }
385         } else {
386 @>              return periodUnitNav[period - 1];
387         }
388     }
```

Even though `settlementPeriod` starts from 1, users can still call `getUnitNav(0)` externally, which will read from `periodUnitNav[uint128(-1)]` leading to an arithmetic underflow error.

Recommendation

Add explicit input validation to ensure the period is always >= 1 before reading from the mapping.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-07 | UNHANDLED ETH IN `onDepositUnderlying` FUNCTION IN `sUSD1PlusVault` AND `SimpleVault` CONTRACTS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/vaults/USD1/sUSD1PlusVault.sol (init): 105~124	Partially Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/sUSD1PlusVault.sol`
- `contracts/vaults/Vault.sol`

A critical vulnerability exists in the `sUSD1PlusVault` contract where the `deposit()` function is marked as payable but the `onDepositUnderlying` function does not properly handle ETH (`msg.value`) sent with the transaction. This results in any ETH sent being permanently locked in the contract with no recovery mechanism.

The vulnerable `deposit()` function is inherited from the `Vault` parent contract:

`contracts/vaults/Vault.sol`

```
237
// @dev deposit underlying asset to the vault, mint shares to the target address
238     function deposit(uint256 underlyingAmount)
239         external
240         whenNotPaused
241         notBlacklisted
242         lock
243         payable
244     {
245         require(underlyingAmount >= minDepositAmount,
"deposit amount too small");
246
247         (uint256 sharesAmount, uint256 unitNav) = onDepositUnderlying(msg.
sender, underlyingAmount);
248
249         // emit enough information for accounting
250         address[] memory p = portfolios();
251         uint256[] memory w = weights();
252         emit Deposited(
253             msg.sender,
254             underlyingAmount,
255             sharesAmount,
256             unitNav,
257             settlementPeriod,
258             p,
259             w,
260             block.timestamp
261         );
262     }
```

The function is marked payable in the parent contract and calls `onDepositUnderlying` hook function. The `sUSD1PlusVault` implements the `onDepositUnderlying` hook function:

`contracts/vaults/USD1/sUSD1PlusVault.sol`

```
105  // to handle the depoisted underlying asset
106      function onDepositUnderlying(address from, uint256 underlyingAmount)
107          internal
108          override
109          returns (uint256 sharesAmount, uint256 unitNav)
110      {
111          // transfer the underlying asset to the ceff wallet
112          address ceffWallet = _portfolios.at(0);
113          IERC20 underlyingToken = IERC20(underlying);
114          SafeERC20.safeTransferFrom(underlyingToken, from, ceffWallet,
 underlyingAmount);
115
116          // calculate the shares amount
117          unitNav = getUnitNav(settlementPeriod);
118          sharesAmount = underlyingAmount * Precision / unitNav;
119
120          // calculate the pending shares
121          pendingShares[from][settlementPeriod] += sharesAmount;
122
123          // emit the event
124          emit PendingShares(from, settlementPeriod, sharesAmount);
125      }
```

The issue here is that the `onDepositUnderlying()` implementation in `sUSD1PlusVault` only processes ERC20 token transfers. It does not check whether `msg.value` is 0. As a result, any ETH sent to the contract via the deposit() function becomes irrecoverable.

The same issue exists in `SimpleVault` contract.

Recommendation

Consider adding following checks in `onDepositUnderlying` in `sUSD1PlusVault` and `SimpleVault` contracts:

```
require(msg.value == 0,"invalid ether value");
```

Alleviation

[Lorenzo, 07/25/2025]: The team heeded the advice and partially resolved the issue by adding checks in `onDepositUnderlying` in `sUSD1PlusVault` contract in commit fd93c26740dfed3f2731456f6209a960373ca495.

LPA-15 `freezeShares()` USES `balanceOf()` INSTEAD OF `getUsableShares()` , POTENTIALLY OVER-FREEZING INACCESSIBLE SHARES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/vaults/Vault.sol (init): 121	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/Vault.sol`

The `freezeShares()` function determines how many shares to freeze by checking `balanceOf(account)` , which includes frozen shares and shares already involved in pending withdrawal requests.

```
120     function freezeShares(address account, uint256 shares) external
onlyManager {
121  @>     uint256 leftShares = balanceOf(account);
122         if (shares > leftShares) {
123             shares = leftShares;
124         }
125
126         freezedShares[account] += shares;
127         emit FreezeShares(account, shares);
128     }
```

This approach may result in double freezing of shares that are already locked due to ongoing withdrawal requests `getOngoingWithdrawShares()` , or prior `freezeShares()` calls.

As a result, `getUsableShares()` may return an incorrect value (possibly 0), even though the user has some actually usable shares, leading to unexpected behavior or user lockout.

Since `getUsableShares()` is used to enforce share availability across critical paths, such as:

- transfer() — checks if sender has enough usable shares

- transferFrom() — same as above for delegated transfers
- requestWithdraw() — ensures sufficient shares are available for withdrawal
- freezeShares() — itself incorrectly uses balanceOf() instead of getUsableShares()

This flawed logic leads to widespread usability problems across the system—for example, users may be unable to transfer shares even though their balance appears sufficient, withdrawal requests might fail unexpectedly due to incorrect availability checks, and frozen shares could exceed the actual free shares, effectively preventing users from accessing their assets.

Recommendation

Consider replace the use of `balanceOf(account)` with `getUsableShares(account)` in the `freezeShares()` function.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue by replacing the use of `balanceOf(account)` with `getUsableShares(account)` in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-17 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/vaults/SimpleVault.sol (init): 28, 36; contracts/vaults/USD1/sUSD1PlusVault.sol (init): 96	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/sUSD1PlusVault.sol`
- `contracts/vaults/SimpleVault.sol`

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
36          cedefiManager = cedefiManager_;
```

- `cedefiManager_` is not zero-checked before being used.

```
 96      function setSigner(address[] memory signers_, bool[] memory isSigners_)
public onlyManager {
 97          require(signers_.length == isSigners_.length,
"mismatch length of signers");
 98
 99          for (uint256 i = 0; i < signers_.length; i++) {
100              signers[signers_[i]] = isSigners_[i];
101          }
102      }
```

- `signers_` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue by adding zero-check in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-08 SELF-ADMINISTERED ROLE ALLOWS FOR COMPLETE ROLE TAKEOVER

Category	Severity	Location	Status
Design Issue	Informational	contracts/CeDeFiManager.sol (init): 28; contracts/vaults/USD1/USD1PlusVault.sol (init): 54	Acknowledged

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/USD1PlusVault.sol`
- `contracts/CeDeFiManager.sol`

In the `initialize` function in `CeDeFiManager`, the `BOSS_ROLE` is assigned as its own admin role, which creates a significant security risk. This setup allows any account with the `BOSS_ROLE` to have administrative control over itself, including the ability to revoke the role from other accounts. **`contracts/CeDeFiManager.sol`**

```
28      _setRoleAdmin(BOSS_ROLE, BOSS_ROLE);
```

If a bad actor gains the `BOSS_ROLE`, they can remove this role from all other accounts, effectively taking complete control.

The same issue exists in the `initialize` function in `USD1PlusVault`, where the `OWNER_ROLE` is assigned as its own admin role.

Recommendation

To mitigate this issue, assign a different, higher-level administrative role to `BOSS_ROLE` / `OWNER_ROLE`. This higher-level role should have the authority to manage the `BOSS_ROLE` / `OWNER_ROLE` assignments, thus preventing any single account with the `BOSS_ROLE` / `OWNER_ROLE` from having unchecked control.

Alleviation

[Lorenzo, 07/24/2025]: The team acknowledged the issue and decided not to implement the recommended change in the

current engagement.

> Yes, it is by design. we should change boss role.

LPA-09 UNNECESSARY `view` MODIFIER IN `_authorizeUpgrade` FUNCTION

Category	Severity	Location	Status
Coding Style	Informational	contracts/CeDeFiManager.sol (init): 37~38	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/CeDeFiManager.sol`

The `_authorizeUpgrade` function in the `CeDeFiManager` contract incorrectly includes a `view` modifier, which may cause issues in future upgrades. While the current implementation (using `onlyRole`) does not modify state, the view restriction is unnecessary and violates the standard UUPS upgrade pattern, potentially leading to unexpected behavior in proxy-based upgrades.

OpenZeppelin's UUPS implementation expects `_authorizeUpgrade` to be a non-view function, as it may need to interact with state (e.g., logging upgrades, dynamic checks):

```
    /**
     * @dev Function that should revert when `msg.sender` is not authorized to
upgrade the contract. Called by
     * {upgradeToAndCall}.
     *
     * Normally, this function will use an xref:access.adoc[access control] modifier
such as {Ownable-onlyOwner}.
     *
     * ```solidity
     * function _authorizeUpgrade(address) internal onlyOwner {}
     * ```
     */
    function _authorizeUpgrade(address newImplementation) internal virtual;
```

The `_authorizeUpgrade` function in `CeDeFiManager` contract uses an unnecessary `view`:

`contracts/CeDeFiManager.sol`

```
37  function _authorizeUpgrade(address newImplementation) internal view override
onlyRole(BOSS_ROLE){
38      }
```

The `view` modifier enforces read-only behavior, which limits future extensibility.

Recommendation

Consider removing the `view` modifier.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue by removing the `view` modifier in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-10 POTENTIAL HASH COLLISION IN `createSimpleVault` 'S SALT CALCULATION

Category	Severity	Location	Status
Coding Issue	Informational	contracts/CeDeFiManager.sol (init): 105~109	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/CeDeFiManager.sol`

The `createSimpleVault` function in the `CeDeFiManager` contract currently uses `abi.encodePacked` to generate a `salt` for vault creation, which carries a risk of hash collisions due to ambiguous parameter packing.

The current implementation:

```
 105  bytes32 salt = keccak256(abi.encodePacked(params.yieldType, params.name, params
.symbol, params.underlying));
```

The `abi.encodePacked` concatenates raw bytes **without** type information or length prefixes. Different parameter combinations can produce identical byte sequences. Consider two vault creation attempts:

Case 1:

```
{
    yieldType: YieldType.Custody, // Assume value = 1
    name: "Token",
    symbol: "Flow",
    underlying: 0x123...
}
```

Case 2:

```
{
    yieldType: YieldType.Custody, // Value = 1
    name: "TokenFlow",
    symbol: "",
    underlying: 0x123...
}
```

With encodePacked, both cases could produce identical salt values because:

1. The enum value (1) and address packing might align similarly
2. String concatenation would be ambiguous without delimiters

Recommendation

Consider using `abi.encode` instead.

Alleviation

[Lorenzo, 07/24/2025]: The team heeded the advice and resolved the issue by using `abi.encode` in commit 32e178c1408f18e49188924b8b28741e9ae5517d.

LPA-11 INCONSISTENT PORTFOLIO ACCOUNTING BETWEEN DEPOSIT AND WITHDRAWAL

Category	Severity	Location	Status
Inconsistency	Informational	contracts/vaults/SimpleVault.sol (init): 48, 80	Acknowledged

Description

In the `SimpleVault` contract, during deposit, the underlying assets are distributed across multiple portfolio addresses based on predefined weights. This behavior is implemented in the `onDepositUnderlying` function:

```
for (uint256 i; i < len;) {
    address portfolio = portfolios[i];
    uint256 weight = portfolioWeights[portfolio];
    uint256 splitUnderlying = (underlyingAmount * weight) / Precision;

    sendUnderlying(portfolio, splitUnderlying);
    ...
}
```

However, in the `onWithdrawUnderlying` function, withdrawals do not reverse this process — instead of recalling funds from the portfolios, the vault sends assets directly to the user from its own balance:

```
  function onWithdrawUnderlying(address to, uint256 underlyingAmount) internal
override {
      sendUnderlying(to, underlyingAmount);
  }
```

The audit team would like to confirm with the project team whether this withdrawal logic — which does not retrieve funds from portfolios during withdrawal — is aligned with the original design intent.

Recommendation

Alleviation

[Lorenzo, 07/24/2025]: Yes, it is designed. the portfolios address just receive tokens and do KYT/KYA for each deposit and we will transfer underlying tokens to the vault contract for withdraws.

LPA-16 MISSING LENGTH VALIDATION IN `updatePortfolios()` MAY CAUSE OUT-OF-BOUNDS ACCESS

Category	Severity	Location	Status
Volatile Code	Informational	contracts/vaults/Vault.sol (init): 44	Resolved

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/Vault.sol`

The `updatePortfolios()` function updates the vault's portfolio addresses and their corresponding weights. However, it does not validate that the input arrays `portfolios_` and `weights_` have the same length.

This can lead to out-of-bounds read errors when accessing `weights_[i]` in the loop, potentially causing a transaction revert.

Recommendation

Add an explicit check at the start of the function to validate that both arrays have the same length.

Alleviation

[Lorenzo, 07/25/2025]: The team heeded the advice and resolved the issue in commit fd93c26740dfed3f2731456f6209a960373ca495.

LPA-19 | MISSING INTERFACE IMPLEMENTATIONE

Category	Severity	Location	Status
Coding Issue	Informational	contracts/test/TestERC20.sol (init): 6~14; contracts/vaults/USD1/IUSD1PlusVault.sol (init): 4~7; contracts/vaults/USD1/USD1PlusVault.sol (init): 15~179	Acknowledged

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/vaults/USD1/USD1PlusVault.sol`

The contract `USD1PlusVault` does not implement the interface `IUSD1PlusVault`, which reduces readability. (Internal use only: it is advised that the auditor checks if the contract implements all funcitons in the interface)

`USD1PlusVault` implements the interface `IUSD1PlusVault`, but does not inherit from it.

```
15  contract USD1PlusVault is
```

```
4  interface IUSD1PlusVault {
```

Recommendation

It is advised to implement the missing interface in the contract to ensure proper functionality and increase readability.

Alleviation

[Lorenzo, 07/24/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> Issue acknowledged. I won't make any changes for the current version.

OPTIMIZATIONS | LORENZO PROTOCOL - AUDIT

ID	Title	Category	Severity	Status
LPA-01	Redundant `If-Else` Statement	Logical Issue	Optimization	Acknowledged

LPA-01 | REDUNDANT `If-Else` STATEMENT

Category	Severity	Location	Status
Logical Issue	Optimization	contracts/CeDeFiManager.sol (init): 137~145, 139~145, 141~145	Acknowledged

Description

Repository:

- `Lorenzo Protocal`

Commit hash:

- `0fd0b768537e52951bdf81aa71d33132e9615eac`

Files:

- `contracts/CeDeFiManager.sol`

The current implementation of the `if-else` clause in `createVault` function is redundant and may lead to confusion in the future:

`contracts/CeDeFiManager.sol`

```
137  else if (params.yieldType == YieldType.PrimeWallet) {
138              // create prime wallet vault
139          } else if (params.yieldType == YieldType.DefiProtocol) {
140              // create defi protocol vault
141          } else if (params.yieldType == YieldType.FromFund) {
142              // create centralized fund vault
143          }
```

It can be simplified to a single `if` statement, making the code clearer and more straightforward.

Recommendation

To mitigate this issue, the current `if-else` clause could be simplified to a single `if` clause.

Alleviation

[Lorenzo, 07/24/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> Issue acknowledged. I won't make any changes for the current version.

APPENDIX | LORENZO PROTOCOL - AUDIT

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.