BLOCKSEC

Security Audit
Report for CoboTokenization

Date: April 1, 2026 **Version:** 1.1

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Cobo
Target	CoboTokenization

Version History

Version	Date	Description
1.0	March 17, 2026	First release
1.1	April 1, 2026	Removed whitelist transfer restriction

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-04-01 07:39:40+0000

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of CoboTokenization of Cobo.

CoboTokenization consists of two upgradeable systems, CoboERC20 and Fund. The CoboERC20 system includes contracts `CoboERC20` and `CoboERC20Wrapper`, where `CoboERC20` provides controlled ERC-20 issuance and transfer controls and `CoboERC20Wrapper` handles underlying asset wrapping and unwrapping with reconciliation-based recovery minting. The Fund system includes contracts `CoboFundOracle`, `CoboFundToken`, and `CoboFundVault`, which maintain APR and compute NAV, manage share subscriptions and asynchronous redemptions, and execute custody settlement payouts. The overall security model relies on role separation and whitelist and blacklist checks on critical execution paths.

Note this audit only focuses on the smart contracts in the following directories/files:

- evm/src

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
cobo-tokenization	`Version 1`	`1b36d0764c85a6d2921d15684d8095598a66986c`
	`Version 2`	`d88ece27960392d8c5081f7edd30c0cc721a3d74`
	`Version 3`	`eba4f58d30a4760300f896d83d9e601188672b44`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/CoboGlobal/cobo-tokenization

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism

* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we have **nine** recommendations and **fourteen** notes.

- Recommendation: 9
- Note: 14

ID	Severity	Description	Category	Status
1	-	Align `minUpdateInterval` validation in function `initialize()`	Recommendation	Fixed
2	-	Use `Math.mulDiv()` for NAV calculation	Recommendation	Fixed
3	-	Add consistency checks in `setOracle()` and `setVault()`	Recommendation	Fixed
4	-	Add non-zero address checks	Recommendation	Fixed
5	-	Remove redundant code	Recommendation	Fixed
6	-	Prevent precision loss in function `getLatestPrice()`	Recommendation	Fixed
7	-	Enforce a lower bound for `minUpdateInterval`	Recommendation	Confirmed
8	-	Align the comment for `pause()` with the implementation	Recommendation	Fixed
9	-	Add minimum validation in function `setMaxAPR()`	Recommendation	Fixed
10	-	Hardcoded external address	Note	-
11	-	Behavior when an account exists in both `_accessList` and `_blockList`	Note	-
12	-	Access list disabled by default	Note	-
13	-	Pausable functionality implementation	Note	-
14	-	Ensure `CoboFundVault` maintains sufficient asset liquidity	Note	-
15	-	Ensure `approve()` policy matches authorization controls	Note	-
16	-	Weird ERC20 tokens	Note	-
17	-	Proxy deployment and implementation binding should be atomic	Note	-
18	-	Potential centralization risks	Note	-
19	-	Access control design for functions `deposit()` and `withdraw()`	Note	-
20	-	Dust shares due to precision loss	Note	-
21	-	Expected migration flow for `setVault()`	Note	-
22	-	Limited burn functionality in contract `CoboERC20Wrapper`	Note	-

23	-	Whitelist restriction removed on transfers in contract `CoboFundToken`	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Align `minUpdateInterval` validation in function `initialize()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `CoboFundOracle`, the function `initialize()` assigns the variable `minUpdateInterval` from the input `_minUpdateInterval` without enforcing the 90-day upper bound that is enforced by the function `setMinUpdateInterval()`. Consequently, `minUpdateInterval` can be initialized to a value greater than 90 days, even though subsequent updates via function `setMinUpdateInterval()` would revert with the error `LibFundErrors.IntervalTooLarge`. This inconsistency permits an out-of-policy configuration at deployment time and may delay oracle updates beyond the intended operational constraints.

```
116         _grantRole(DEFAULT_ADMIN_ROLE, admin);
117
118         baseNetValue = initialNetValue;
119         currentAPR = initialAPR;
120         lastUpdateTimestamp = block.timestamp;
121         maxAPR = _maxAPR;
122         maxAprDelta = _maxAprDelta;
123         minUpdateInterval = _minUpdateInterval;
124     }
```

Listing 2.1: evm/src/Fund/CoboFundOracle.sol

```
181     function setMinUpdateInterval(uint256 _minUpdateInterval) external onlyRole(DEFAULT_ADMIN_ROLE
            ) {
182         if (_minUpdateInterval > 90 days) revert LibFundErrors.IntervalTooLarge(_minUpdateInterval,
                90 days);
183         minUpdateInterval = _minUpdateInterval;
184         emit MinUpdateIntervalUpdated(_minUpdateInterval);
185     }
```

Listing 2.2: evm/src/Fund/CoboFundOracle.sol

Suggestion Add the same 90-day upper-bound check in the function `initialize()` for the input `_minUpdateInterval`.

Clarification from BlockSec The inconsistency has been resolved by removing the 90-day upper bound from the function `setMinUpdateInterval()`, allowing `minUpdateInterval` to be set as an arbitrarily large value. To avoid delaying APR updates, the project should exercise caution when configuring this parameter.

2.1.2 Use `Math.mulDiv()` for NAV calculation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the project, function `getLatestPrice()` of contract `CoboFundOracle` and functions `_assetToShares()` and `_sharesToAsset()` of contract `CoboFundToken` execute multiplication-then-division arithmetic, which can lead to overflow during multiplication. Using `Math.mulDiv()` would improve arithmetic precision and reduce the risk of intermediate overflow during the calculation.

```
129   function getLatestPrice() public view override returns (uint256) {
130       uint256 elapsed = block.timestamp - lastUpdateTimestamp;
131       // Optimized calculation to reduce overflow risk:
132       // growth = (baseNetValue * currentAPR / _PRECISION) * elapsed / _SECONDS_PER_YEAR
133       uint256 annualGrowth = (baseNetValue * currentAPR) / _PRECISION;
134       return baseNetValue + (annualGrowth * elapsed) / _SECONDS_PER_YEAR;
135   }
```

Listing 2.3: evm/src/Fund/CoboFundOracle.sol

```
227   function _assetToShares(uint256 assetAmount, uint256 nav) internal view returns (uint256) {
228       return (assetAmount * _shareScale * _PRECISION) / (nav * _assetScale);
229   }
```

Listing 2.4: evm/src/Fund/CoboFundToken.sol

Suggestion Use OpenZeppelin's `Math.mulDiv()` for the NAV calculation in function `getLatestPrice()`.

2.1.3 Add consistency checks in `setOracle()` and `setVault()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `CoboFundToken`, the functions `setOracle()` and `setVault()` replace critical endpoints. However, the functions lack validation on the new endpoints to ensure state continuity.

For the `oracle` migration path, the new `oracle`'s state variables `baseNetValue`, `currentAPR`, and `lastUpdateTimestamp` are expected to remain coherent across the switch.

For the `vault` migration path, the new `vault` should grant maximum allowances and have sufficient asset balances to honor outstanding redemption liabilities.

```
409   /// @notice Set the NAV oracle address.
410   function setOracle(address oracle_) external onlyRole(DEFAULT_ADMIN_ROLE) {
411       if (oracle_ == address(0)) revert LibFundErrors.ZeroAddress();
412       oracle = ICoboFundOracle(oracle_);
413       emit OracleUpdated(oracle_);
414   }
415
416   /// @notice Set the asset custody vault address.
417   function setVault(address vault_) external onlyRole(DEFAULT_ADMIN_ROLE) {
418       if (vault_ == address(0)) revert LibFundErrors.ZeroAddress();
```

```
419         vault = vault_;
420         emit VaultUpdated(vault_);
421     }
```

Listing 2.5: evm/src/Fund/CoboFundToken.sol

Suggestion Add checks to the functions `setOracle()` and `setVault()` before updating the endpoints.

2.1.4 Add non-zero address checks

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `deployAndInit()`, the variables `managers[i]`, `minters[i]`, `burners[i]`, `pausers[i]`, `salvagers[i]`, and `upgraders[i]` are not validated against `address(0)` prior to invoking the function `grantRole()` for role assignment. Furthermore, in the function `grantRole()`, the variable `account` is not checked to ensure it is non-zero. This omission may lead to inadvertent role assignment to the zero address, increasing the likelihood of operational misconfiguration.

```
64          for (uint256 i = 0; i < admins.length; i++) {
65              // check if admin is empty
66              if (admins[i] == address(0)) revert InvalidAddress();
67              coboERC20Proxy.grantRole(coboERC20Proxy.DEFAULT_ADMIN_ROLE(), admins[i]);
68          }
69          for (uint256 i = 0; i < managers.length; i++) {
70              coboERC20Proxy.grantRole(coboERC20Proxy.MANAGER_ROLE(), managers[i]);
71          }
72          for (uint256 i = 0; i < minters.length; i++) {
73              coboERC20Proxy.grantRole(coboERC20Proxy.MINTER_ROLE(), minters[i]);
74          }
75          for (uint256 i = 0; i < burners.length; i++) {
76              coboERC20Proxy.grantRole(coboERC20Proxy.BURNER_ROLE(), burners[i]);
77          }
78          for (uint256 i = 0; i < pausers.length; i++) {
79              coboERC20Proxy.grantRole(coboERC20Proxy.PAUSER_ROLE(), pausers[i]);
80          }
81          for (uint256 i = 0; i < salvagers.length; i++) {
82              coboERC20Proxy.grantRole(coboERC20Proxy.SALVAGER_ROLE(), salvagers[i]);
83          }
84          for (uint256 i = 0; i < upgraders.length; i++) {
85              coboERC20Proxy.grantRole(coboERC20Proxy.UPGRADER_ROLE(), upgraders[i]);
86          }
87          coboERC20Proxy.renounceRole(coboERC20Proxy.DEFAULT_ADMIN_ROLE(), _this);
```

Listing 2.6: evm/src/deploy/ProxyFactory.sol

Suggestion Add explicit non-zero address checks for `managers[i]`, `minters[i]`, `burners[i]`, `pausers[i]`, `salvagers[i]`, and `upgraders[i]` in the function `deployAndInit()` before calling the function `grantRole()`, and add a non-zero validation for the variable `account` in the function `grantRole()` to prevent role grants to `address(0)`.

2.1.5 Remove redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There are several unused variables, events, and functions. It is recommended to remove them for better code readability. Specifically, the following code should be removed or revised.

1. In the library `LibFundErrors`, the error `InsufficientBalance` is currently unused in the source path.
2. In the contract `ProxyFactory`, the remaining `TODO` comments appear in executable deployment logic and should be cleaned up or replaced with finalized implementation notes.

```
24    error InsufficientBalance(address user, uint256 requested, uint256 available);
```

Listing 2.7: evm/src/Fund/libraries/LibFundErrors.sol

```
52        // TODO: add init code
53        address proxy = factory.doDeploy(
54            finalSalt,
55            abi.encodePacked(
56                type(ERC1967Proxy).creationCode,
57                abi.encode(coboERC20Logic,bytes(""))
58            )
59        );
60        CoboERC20 coboERC20Proxy = CoboERC20(proxy);
61        // TODO: initialize
62        coboERC20Proxy.initialize(name, symbol, uri, decimal, _this);
63        // TODO: add admin, manager, minter, burner, pauser, salvager, upgrader
```

Listing 2.8: evm/src/deploy/ProxyFactory.sol

Suggestion Remove the redundant code.

2.1.6 Prevent precision loss in function `getLatestPrice()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `CoboFundOracle`, function `getLatestPrice()` is intended to compute linear NAV growth from `baseNetValue`, `currentAPR`, and elapsed time, but the current implementation divides by `_PRECISION` before applying `elapsed`, which truncates fractional growth and introduces avoidable precision loss.

```
129   function getLatestPrice() public view override returns (uint256) {
130       uint256 elapsed = block.timestamp - lastUpdateTimestamp;
131       // Optimized calculation to reduce overflow risk:
132       // growth = (baseNetValue * currentAPR / _PRECISION) * elapsed / _SECONDS_PER_YEAR
133       uint256 annualGrowth = (baseNetValue * currentAPR) / _PRECISION;
134       return baseNetValue + (annualGrowth * elapsed) / _SECONDS_PER_YEAR;
135   }
```

Listing 2.9: evm/src/Fund/CoboFundOracle.sol

Suggestion Reorder the operations to multiply before dividing.

2.1.7 Enforce a lower bound for `minUpdateInterval`

Status Confirmed

Introduced by `Version 1`

Description In contract `CoboFundOracle`, the state variable `minUpdateInterval` is intended to enforce a minimum time gap between APR updates. However, the contract does not enforce any lower bound on this configuration. When the variable `minUpdateInterval` is set to zero, APR updates can be executed without any effective rate limiting.

```
100   function initialize(
101       address admin,
102       uint256 initialNetValue,
103       uint256 initialAPR,
104       uint256 _maxAPR,
105       uint256 _maxAprDelta,
106       uint256 _minUpdateInterval
107   ) external initializer {
108       if (admin == address(0)) revert LibFundErrors.ZeroAddress();
109       if (initialNetValue == 0) revert LibFundErrors.ZeroNetValue();
110       if (initialAPR > _maxAPR) revert LibFundErrors.APRExceedsMax(initialAPR, _maxAPR);
111
112       __AccessControlEnumerable_init();
113       __Multicall_init();
114       __UUPSUpgradeable_init();
115
116       _grantRole(DEFAULT_ADMIN_ROLE, admin);
117
118       baseNetValue = initialNetValue;
119       currentAPR = initialAPR;
120       lastUpdateTimestamp = block.timestamp;
121       maxAPR = _maxAPR;
122       maxAprDelta = _maxAprDelta;
123       minUpdateInterval = _minUpdateInterval;
124   }
```

Listing 2.10: evm/src/Fund/CoboFundOracle.sol

```
181   function setMinUpdateInterval(uint256 _minUpdateInterval) external onlyRole(DEFAULT_ADMIN_ROLE
          ) {
182       if (_minUpdateInterval > 90 days) revert LibFundErrors.IntervalTooLarge(_minUpdateInterval,
              90 days);
183       minUpdateInterval = _minUpdateInterval;
184       emit MinUpdateIntervalUpdated(_minUpdateInterval);
185   }
```

Listing 2.11: evm/src/Fund/CoboFundOracle.sol

Suggestion Add a lower bound for the `minUpdateInterval` in the functions `initialize()` and `setMinUpdateInterval()`.

Feedback from the project The project confirmed this is an intentional design. It preserves flexibility for emergency NAV corrections, while `maxAprDelta` already limits the magnitude of any single APR change. `NAV_UPDATER_ROLE` is managed via Cobo Guard policies, and a sensible minimum interval will be configured via `setMinUpdateInterval()` at deployment time.

2.1.8 Align the comment for `pause()` with the implementation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `PauseUpgradeable`, the NatSpec comment (line 32) for the function `pause()` states that the caller must hold `MANAGER_ROLE`, even though the access control check is delegated to the function `_authorizePause()`.

```
27 /**
28  * @notice This is a function used to pause the contract.
29  *
30  * @dev Calling Conditions:
31  *
32  * - The caller must hold the "MANAGER_ROLE" role.
33  *
34  * This function emits a {Paused} event as part of {PausableUpgradeable._pause}.
35  */
36 function pause() external virtual {
37   _authorizePause();
38   _pause();
39 }
```

Listing 2.12: evm/src/CoboERC20/library/Utils/PauseUpgradeable.sol

```
351   /**
352    * @notice This is a function that applies any validations required to allow pause operations
           to be executed.
353    *
354    * @dev Reverts when the caller does not have the "PAUSER_ROLE".
355    *
356    * Calling Conditions:
357    *
358    * - Only the "PAUSER_ROLE" can execute.
359    */
360   /* solhint-disable no-empty-blocks */
361   function _authorizePause() internal virtual override onlyRole(PAUSER_ROLE) {}
```

Listing 2.13: evm/src/CoboERC20/CoboERC20.sol

Suggestion Consider updating the comment to reflect the actual required role, or revising the authorization logic if `MANAGER_ROLE` was intended.

2.1.9 Add minimum validation in function `setMaxAPR()`

Status Fixed in `Version 2`

Introduced by Version 1

Description In contract CoboFundOracle, function setMaxAPR() configures maxAPR and does not currently enforce this constraint. The annotation in the test file CoboFundOracle.t.sol, "Production code prevents setMaxAPR below currentAPR", indicates that the maxAPR should be greater than currentAPR. However, the function does not currently enforce this constraint.

```
169    function setMaxAPR(uint256 _maxAPR) external onlyRole(DEFAULT_ADMIN_ROLE) {
170        maxAPR = _maxAPR;
171        emit MaxAPRUpdated(_maxAPR);
172    }
```

Listing 2.14: evm/src/Fund/CoboFundOracle.sol

Suggestion Add a validation to ensure that the APR configuration is consistent with the current Oracle state.

2.2 Note

2.2.1 Hardcoded external address

Introduced by Version 1

Description In contract ProxyFactory, function deployAndInit() relies on a fixed external factory address to create new proxy instances. This deployment flow is safe under the assumption that the factory is available in every intended deployment chain and consistently exhibits the expected behavior.

```
46        if (admins.length == 0) revert InvalidAddress();
47
48        address _this = address(this);
49        IFactory factory = IFactory(0xC0B000003148E9c3E0D314f3dB327Ef03ADF8Ba7);
```

Listing 2.15: evm/src/deploy/ProxyFactory.sol

Feedback from the project Address 0xC0B000003148E9c3E0D314f3dB327Ef03ADF8Ba7 is a proprietary deterministic deployment infrastructure deployed consistently across all target chains. This dependency will be documented in the deployment specification.

2.2.2 Behavior when an account exists in both _accessList and _blockList

Introduced by Version 1

Description In the contract CoboERC20, the function _requireAccess() checks if an account has access permissions. If an account exists in both the _accessList and _blockList, the function _requireAccess() will revert due to the _blockList check.

```
403    function _requireAccess(address account) internal view virtual {
404        if (accessListEnabled) {
405            if (!_accessList.contains(account)) revert LibErrors.NotAccessListAddress(account);
406        }
407
```

```
408         if (_blockList.contains(account)) revert LibErrors.BlockedAddress(account);
409     }
```

Listing 2.16: evm/src/CoboERC20/CoboERC20.sol

Feedback from the project The project confirms that `blockList` takes precedence over `accessList`, where a blocked address is denied even if it is also on the access list. This behavior is consistent with compliance intent (regulatory sanctions override access permissions), and will be documented.

2.2.3 Access list disabled by default

Introduced by `Version 1`

Description The function `__AccessList_init()` in the contract `AccessListUpgradeable` initializes variable `accessListEnabled` as false by default. This means that there is always a time window between contract deployment and the explicit enabling of the access list, during which users not on the access list can still perform transfers or other operations.

```
92 function __AccessList_init() internal virtual onlyInitializing {
93   accessListEnabled = false;
94 }
```

Listing 2.17: evm/src/CoboERC20/library/Utils/AccessListUpgradeable.sol

Feedback from the project The project confirmed this is an intentional design.

2.2.4 Pausable functionality implementation

Introduced by `Version 1`

Description The contract `CoboERC20` uses OpenZeppelin's `PauseUpgradeable`, but does not apply the modifier `whenNotPaused` to the functions `burn()` and `burnFrom()`. This means that burning operations remain unprotected when the contract is paused.

```
194     function burn(uint256 amount) external virtual onlyRole(BURNER_ROLE) {
195         if (amount == 0) revert LibErrors.ZeroAmount();
196         _burn(_msgSender(), amount);
197     }
```

Listing 2.18: evm/src/CoboERC20/CoboERC20.sol

```
214     function burnFrom(address account, uint256 amount) external virtual onlyRole(MANAGER_ROLE) {
215         if (amount == 0) revert LibErrors.ZeroAmount();
216         _burn(account, amount);
217     }
```

Listing 2.19: evm/src/CoboERC20/CoboERC20.sol

Feedback from the project The project confirmed this is an intentional design.

2.2.5 Ensure `CoboFundVault` maintains sufficient asset liquidity

Introduced by `Version 1`

Description In the contract `CoboFundToken`, the function `approveRedemption()` finalizes redemptions by transferring asset tokens from the contract `CoboFundVault` via `asset.safeTransferFrom(vault, user, assetAmount)`. Operational procedures should ensure that the contract `CoboFundVault` continuously maintains an adequate asset-token balance, so that pending redemptions remain executable under normal and stressed conditions. Furthermore, balances should be monitored and replenished in advance of redemption approvals to avoid unexpected settlement reverts during execution.

2.2.6 Ensure `approve()` policy matches authorization controls

Introduced by `Version 1`

Description In contracts `CoboERC20`, `CoboERC20Wrapper`, and `CoboFundToken`, the function `approve()` follows the standard ERC20 model and is not subject to the role-based permissions, list-based restrictions, or pause constraints enforced on transfer paths.

The project should confirm that this broader authorization boundary is intentional, document the distinction, and ensure downstream components relying on allowances (including wrappers, spenders, or automation) are designed with the same governance and pause assumptions applied to transfer flows.

Feedback from the project The project confirmed this is an intentional design.

2.2.7 Weird ERC20 tokens

Introduced by `Version 1`

Description In this project, it is important to note that some weird ERC20 tokens (e.g., Fee-on-Transfer tokens, rebasing tokens) are not supported and may result in unexpected behaviors. It is important to note that some weird ERC20 tokens (e.g., Fee-on-Transfer tokens) may result in unexpected behaviors. For instance, the `underlying` token for `CoboERC20Wrapper` and the `asset` token used by `CoboFundVault` are expected to behave like standard ERC20 tokens that maintain a 1:1 relationship between transferred amounts and wrapped token supply. If these tokens apply transfer fees or rebased balances, minting, redemption, and accounting can deviate from intended results.

Feedback from the project Only standard ERC-20 tokens are supported (fixed balances, no transfer fees, no rebase mechanism). Fee-on-transfer tokens cause the vault to receive less than the minted share value, and rebasing tokens decouple vault balance changes from the share ledger.

2.2.8 Proxy deployment and implementation binding should be atomic

Introduced by `Version 1`

Description To prevent potential front-running attacks, it is recommended that proxy deployment and implementation binding be executed atomically within a single transaction. If

these operations are performed separately, malicious actors could exploit the time window between deployment and binding to front-run the implementation binding transaction. An attacker could potentially bind their own malicious implementation to the newly deployed proxy contract, gaining unauthorized control over the protocol's upgrade mechanism and user funds. This race condition poses significant security risks, including complete protocol compromise, fund theft, and unauthorized access to privileged functions.

Feedback from the project In `ProxyFactory.deployAndInit()`, proxy deployment (`factory.doDeploy()`) and initialization (`coboERC20Proxy.initialize()`) are executed sequentially within the same transaction, providing inherent atomicity with no front-running window.

Clarification from BlockSec While the current codebase specifically references the deployment pattern for `coboERC20Proxy`, the same atomic deployment principle applies universally. We assume all other contracts within the system follow the same secure deployment pattern as described in the project's feedback.

2.2.9 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`, `PAUSER_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, privileged roles can upgrade implementations, pause or unpause the system, mint or burn tokens, manage access lists, approve or reject redemptions, force redeems, rescue tokens, and update core configuration parameters. Beyond these on-chain configurations, the protocol also extends implicit trust to `NAV_UPDATER_ROLE`, assuming it always supplies accurate and reasonable off-chain NAV inputs. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.2.10 Access control design for functions `deposit()` and `withdraw()`

Introduced by `Version 1`

Description The contract `CoboERC20Wrapper` implements two access control mechanisms for the underlying tokens. First, functions `deposit()` and `withdraw()` are restricted to `WRAPPER_ROLE`. Second, functions `transfer()` and `transferFrom()` enforce all participating users to pass the `_requireAccess()` validation.

```
308    function deposit(uint256 value) public virtual whenNotPaused onlyRole(WRAPPER_ROLE) returns (
           bool) {
309        address sender = _msgSender();
310        IERC20(_underlying).safeTransferFrom(sender, address(this), value);
311        _mint(sender, value);
312        emit Deposit(sender, value);
313        return true;
314    }
315
316    /**
317     * @dev Allow a user to burn a number of wrapped tokens and withdraw the corresponding number
              of underlying tokens.
```

```
318     */
319     function withdraw(uint256 value) public virtual whenNotPaused onlyRole(WRAPPER_ROLE) returns (
            bool) {
320         address sender = _msgSender();
321         _burn(sender, value);
322         IERC20(_underlying).safeTransfer(sender, value);
323         emit Withdrawal(sender, value);
324         return true;
325     }
```

Listing 2.20: evm/src/CoboERC20/CoboERC20Wrapper.sol

```
250     function transferFrom(
251         address from,
252         address to,
253         uint256 amount
254     ) public virtual override whenNotPaused returns (bool) {
255         _requireAccess(_msgSender());
256         _requireAccess(from);
257         _requireAccess(to);
258 
259         return super.transferFrom(from, to, amount);
260     }
```

Listing 2.21: evm/src/CoboERC20/CoboERC20Wrapper.sol

```
442     function _requireAccess(address account) internal view virtual {
443         if (accessListEnabled) {
444             if (!_accessList.contains(account)) revert LibErrors.NotAccessListAddress(account);
445         }
446 
447         if (_blockList.contains(account)) revert LibErrors.BlockedAddress(account);
448     }
```

Listing 2.22: evm/src/CoboERC20/CoboERC20Wrapper.sol

Feedback from the project The project confirmed this is an intentional design.

2.2.11 Dust shares due to precision loss

Introduced by Version 1

Description In contract `CoboFundOracle`, functions `_assetToShares()` and `_sharesToAsset()` handle conversions between shares and assets that may have different decimal precisions. When the share decimal exceeds the asset decimal, the precision loss in function `_sharesToAsset()` causes the returned asset amount to be lower than expected. As a result, shares may not be fully redeemed for their corresponding underlying assets, leaving irrecoverable dust shares in the protocol.

2.2.12 Expected migration flow for `setVault()`

Introduced by Version 1

Description In the contract `CoboFundToken`, function `setVault()` switches the active vault immediately, while assets already deposited remain in the previous vault. Meanwhile, the function `approveRedemption()` always transfers assets from the current vault. The project should migrate all assets to the new vault off-chain before invoking function `setVault()`.

```
417    function setVault(address vault_) external onlyRole(DEFAULT_ADMIN_ROLE) {
418        if (vault_ == address(0)) revert LibFundErrors.ZeroAddress();
419        vault = vault_;
420        emit VaultUpdated(vault_);
421    }
```

Listing 2.23: evm/src/Fund/CoboFundToken.sol

Feedback from the project The project confirms that the funds will be migrated off-chain before invoking the function `setVault()`.

2.2.13 Limited burn functionality in contract `CoboERC20Wrapper`

Introduced by `Version 1`

Description In contract `CoboERC20Wrapper`, there is no privileged burn functionality equivalent to functions `burnFrom()` and `forceRedeem()` in contracts `CoboERC20` and `CoboFundToken`, which burn tokens from a specified address. The only function to burn wrapper tokens is `withdraw()`, which burns tokens held by `WRAPPER_ROLE` and transfers the underlying tokens to `WRAPPER_ROLE`.

Feedback from the project The project confirmed this is an intentional design. As function `withdraw()` serves as the equivalent destruction path and is restricted to `WRAPPER_ROLE`, the current version does not include a standalone privileged burn interface. If a concrete compliance requirement arises, such an interface can be introduced via a UUPS upgrade.

2.2.14 Whitelist restriction removed on transfers in contract `CoboFundToken`

Introduced by `Version 3`

Description `Version 3` introduces a notable business logic change in contract `CoboFundToken`. Specifically, the whitelist restriction on token transfers is removed, allowing tokens to circulate freely once issued to whitelisted addresses.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS