BLOCKSEC

Security Audit Report for CoboTokenization

Date: August 1, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Cobo
Target	CoboTokenization

Version History

Version	Date	Description
1.0	August 1, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)
SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C
Date: 2025-08-04 09:01:21+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

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

The target of this audit is a **ZIP** archive of CoboTokenization of Cobo.

CoboTokenization is a sophisticated, upgradeable ERC20 token implementation with role-based access control, and can be deployed on different chains with same address via the contract ProxyFactory.

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

- src/*

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope 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 files that fix the discovered issues. If there are new issues, we will continue this process. The MD5 hashes of the audited files 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.

Project	Version	MD5 Hash
CoboTokenization	Version 1	28743f93aa5e695b726313ea960b2a69
	Version 2	e80f2e3b2edc9851b149fcd14ce745c5

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. 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 smart contracts, 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 smart contracts.

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 smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts 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 [1] and Common Weakness Enumeration [2]. 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.

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

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

Chapter 2 Findings

In total, we found **one** potential security issue. Besides, we have **one** recommendation and **four** notes.

- Medium Risk: 1
- Recommendation: 1
- Note: 4

ID	Severity	Description	Category	Status
1	Medium	Potential front-run attack in the function `deployAndInit()`	Security Issue	Fixed
2	-	Add zero address checks	Recommendation	Confirmed
3	-	Access list disabled by default	Note	-
4	-	Behavior when an account exists in both `_accessList` and `_blockList`	Note	-
5	-	Pausable functionality implementation	Note	-
6	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential front-run attack in the function `deployAndInit()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `deployAndInit()` invokes `factory.doDeploy()` to deploy a proxy contract `CoboERC20Proxy`. However, since the `proxy` address is generated via `deployCreate3` (which depends on `msg.sender` and `salt`), and the `msg.sender` in this case is the contract `ProxyFactory`, an attacker could front-run a victim's `deployAndInit()` invocation by submitting an identical transaction with the same `salt`.

This would result in the same proxy address being generated, but the attacker could hijack the deployment by setting critical parameters (e.g., `admins`, `managers`, etc.) to their own addresses, leading to hijacking the token contract.

Moreover, the function `deployAndInit()` allows any caller to deploy a new `CoboERC20` token proxy and initialize it with custom parameters. The `admins` and `managers` (along with other roles like `minters`, `burners`, etc.) can be arbitrarily specified by the caller without any validation or restrictions. This could potentially lead to unauthorized token deployments or privilege escalation if not properly restricted.

```
function deployAndInit(
    uint256 salt, // salt for proxy deployment, eg: uint256(bytes32("CoboERC20Proxy"))
    address coboERC20Logic, // coboERC20 logic address
    string memory name, // name
```

```
        string memory symbol, // symbol
        string memory uri, // uri
        uint8 decimal, // decimal
        address[] memory admins, // admin address
        address[] memory managers, // managers address
        address[] memory minters, // minters address
        address[] memory burners, // burners address
        address[] memory pausers, // pausers address
        address[] memory salvagers, // salvagers address
        address[] memory upgraders // upgraders address
    ) public returns (address) {
        // check if admins is empty
        if (admins.length == 0) revert InvalidAddress();

        address _this = address(this);
        IFactory factory = IFactory(0xC0B000003148E9c3E0D314f3dB327Ef03ADF8Ba7);

        // TODO: add init code
        address proxy = factory.doDeploy(
            salt,
            abi.encodePacked(
                type(ERC1967Proxy).creationCode,
                abi.encode(coboERC20Logic,bytes(""))
            )
        );
        CoboERC20 coboERC20Proxy = CoboERC20(proxy);
        // TODO: initialize
        coboERC20Proxy.initialize(name, symbol, uri, decimal, _this);
        // TODO: add admin, manager, minter, burner, pauser, salvager, upgrader
        for (uint256 i = 0; i < admins.length; i++) {
            // check if admin is empty
            if (admins[i] == address(0)) revert InvalidAddress();
            coboERC20Proxy.grantRole(coboERC20Proxy.DEFAULT_ADMIN_ROLE(), admins[i]);
        }
        for (uint256 i = 0; i < managers.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.MANAGER_ROLE(), managers[i]);
        }
        for (uint256 i = 0; i < minters.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.MINTER_ROLE(), minters[i]);
        }
        for (uint256 i = 0; i < burners.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.BURNER_ROLE(), burners[i]);
        }
        for (uint256 i = 0; i < pausers.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.PAUSER_ROLE(), pausers[i]);
        }
        for (uint256 i = 0; i < salvagers.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.SALVAGER_ROLE(), salvagers[i]);
        }
        for (uint256 i = 0; i < upgraders.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.UPGRADER_ROLE(), upgraders[i]);
        }
        coboERC20Proxy.renounceRole(coboERC20Proxy.DEFAULT_ADMIN_ROLE(), _this);
```

```

        return proxy;
    }
```

Listing 2.1: src/deploy/ProxyFactory.sol

Impact Unauthorized tokens with unexpected parameters can be deployed.

Suggestion Revise the code logic accordingly.

2.2 Recommendation

2.2.1 Add zero address checks

Status Confirmed

Introduced by Version 1

Description In the function `deployAndInit()`, several address variables (e.g., `managers`, `minters`) are not checked to ensure they are not zero. It is recommended to add such checks to prevent potential mis-operations.

```
        for (uint256 i = 0; i < managers.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.MANAGER_ROLE(), managers[i]);
        }
        for (uint256 i = 0; i < minters.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.MINTER_ROLE(), minters[i]);
        }
        for (uint256 i = 0; i < burners.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.BURNER_ROLE(), burners[i]);
        }
        for (uint256 i = 0; i < pausers.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.PAUSER_ROLE(), pausers[i]);
        }
        for (uint256 i = 0; i < salvagers.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.SALVAGER_ROLE(), salvagers[i]);
        }
        for (uint256 i = 0; i < upgraders.length; i++) {
            coboERC20Proxy.grantRole(coboERC20Proxy.UPGRADER_ROLE(), upgraders[i]);
        }
```

Listing 2.2: src/deploy/ProxyFactory.sol

Suggestion Add non-zero address checks accordingly.

2.3 Note

2.3.1 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.

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

Listing 2.3: src/CoboERC20/library/Utils/AccessListUpgradeable.sol

Feedback from the project It is by design.

2.3.2 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.

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

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

Listing 2.4: src/CoboERC20/CoboERC20.sol

Feedback from the project It is by design.

2.3.3 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.

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

Listing 2.5: src/CoboERC20/CoboERC20.sol

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

Listing 2.6: src/CoboERC20/CoboERC20.sol

Feedback from the project It is by design.

2.3.4 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `MINTER_ROLE`, `DEFAULT_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, `MINTER_ROLE` can mint tokens to users based on the protocol. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS