Audit Report

CoMakery Security Token Smart Contracts

December 27, 2021

Version 0.2

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by Republic Crypto LLC to perform a security audit of the CoMakery security token smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/CoMakery/comakery-security-token

Commit hash: `3967919707d082569370015ea9385ccd4ab2e4d8`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The submitted contracts implement a security token in the form of an Ethereum-based ERC-20 token smart contract with additional support for ERC-1404 transfer restrictions.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	A reentrancy in `cancelSell` can be used to drain the contract	Critical	Resolved
2	ERC1404 check returns true for EOAs which could lead to errors	Minor	Resolved
3	README does not match implementation in the `RestrictedToken` constructor	Minor	Resolved
4	Protection to leave the contract without administrator can be bypassed accidentally	Minor	Resolved
5	An address which does not conform to the `ITransferRules` interface passed to `upgradeTransferRules` renders the contract temporarily unusable	Minor	Resolved
6	Untrusted ERC-20 tokens used in RestrictedSwap	Minor	Pending
7	Once the maximum number of timelocks for an address has been reached existing timelocks cannot be updated	Minor	Pending
8	The `swapNumber` function parameter shadows the global `swapNumber` variable	Informational	-
9	`MAXUINT256` is an unused variable	Informational	-
10	`getLockUntilTimestampLookup` and `setAddressPermissions` can be external.	Informational	-
11	Redundant value check in `burn()`	Informational	-
12	uint8s greater than 9 passed to `messageForTransferRestriction` return an empty string	Informational	-
13	Redundant if statement in `messageForTransferRestriction`	Informational	-
14	Reentrancy allows for arbitrary `_swap` writes	Informational	-

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of Documentation	**Medium-high**	The contracts are well-documented with code comments. However, the README file is out of date with the current state of contracts.
Test Coverage	**Medium-High**	`getLockUntilTimestampLookup`, `checkReserveAdmin`, `checkWalletsAdmin`, and lines 537 and 564 of `RestrictedToken` are not covered with tests. The `BuyConfigured` branch of `cancelSell` in the `RestrictedSwap` is not covered with tests.

Detailed Findings

1. A reentrancy in `cancelSell` can be used to drain the contract

Severity: Critical

Function `cancelSell` in `RestrictedSwap` makes a call to an external ERC-20 token contract. However, since the cancel status is updated after the call a malicious token might re-enter the function multiple times and drain the contract of its funds.

Recommendation

We recommend adding a reentrancy guard to all functions with external calls to untrusted ERC-20 tokens in `RestrictedSwap`. An example of such a guard can be found in the OpenZeppelin smart contract library (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol).

Status: Resolved

2. ERC1404 check returns true for EOAs which could lead to errors

Severity: Minor

In file `RestrictedSwap` in the function `_configureSwap` uses the helper function `isERC1404` to check whether an address implements a certain interface. However, in testing, it was found that the EVM returns true for externally owned account addresses as well, which could lead to strange non-descriptive error messages that are hard to trace.

Recommendation

We recommend adding additional checks on the `returnData` returned from the `quoteToken detectTransferRestriction` call.

Status: Resolved

3. The README does not match the implementation in the `RestrictedToken` constructor

Severity: Minor

In L183 of the README it is stated that “The wallet account address holding all of the initially issued tokens should not have any admin roles associated with them.” However, the implementation of the `RestrictedToken` constructor does not match this description.

Recommendation

We recommend updating the constructor implementation to match the README description, or updating the README to reflect the constructor implementation.

Status: Resolved

4. Protection to leave the contract without administrator can be bypassed accidentally

Severity: Minor

The empty address can be passed to `grantContractAdmin` of `RestrictedToken`, allowing the `contractAdminCount` to be changed, bypassing the require on `contractAdminCount` in `revokeContractAdmin`.
The reason for this is tha the OpenZeppellin role system used allows roles to be renounced, which could lead to the invariant defined in line 194 being violated.

Recommendation

We recommend adding a requirement that the address passed to `grantContractAdmin` not be the empty address. Similarly, the OZ `renounceRole` function could be overwritten.

Status: Resolved

5. An address which does not conform to the `ITransferRules` interface passed to `upgradeTransferRules` renders the contract temporarily unusable

Severity: Minor

If an address that does not conform to the `ITransferRules` interface is passed to the `upgradeTransferRules` function of `RestrictedToken`, the contract becomes unusable until fixed by the administrator.

Recommendation

We recommend adding a check that the passed-in address conforms to the `ITransferRules` interface, similar to what was done in L80 of `RestrictedSwap` for the `ERC1404` interface.

Status: Resolved

6. Untrusted ERC-20 tokens used in `RestrictedSwap`

Severity: Minor

External ERC-20 tokens which need to be trusted are used throughout the `RestrictedSwap` contract. This can lead to unintended behaviour when an ERC-20 token does not conform to the expected behaviour. For example, a malicious token might divert a percentage of every transfer.

Recommendation

We recommend implementing a whitelist mechanism such that only trusted ERC-20 tokens can be used in `RestrictedSwap`.

Status: Pending

7. Once the maximum number of timelocks for an address has been reached existing timelocks cannot be updated

Severity: Minor

The function `addLockUntil` in `RestrictedToken` iterates over the existing timelocks, in order to allow for updates, which indicates that allowing locks to be modified is desired behaviour. However, the `require` statements in line 259 will always cause the transaction to revert if the maximum number of allowed locks has been reached, meaning no modifications are possible.

Recommendation

We recommend allowing for modifications by considering the special case of all lock slots having been used or implementing a separate function that allows the modification of a particular timelock.

Status: Pending

8. The `swapNumber` function parameter shadows the global `swapNumber` variable

Severity: Informational

In the `RestrictedSwap.sol` contract, the `swapNumber` function parameter shadows the global `swapNumber` variable.

Recommendation

We recommend changing the name of the `swapNumber` function parameter found throughout the contract.

9. `MAXUINT256` is an unused variable

Severity: Informational

The `MAXUINT256` variable in the `RestrictedSwap` contract is not used.

Recommendation

We recommend removing `MAXUINT256`.

10. `getLockUntilTimestampLookup` and `setAddressPermissions` can be external

Severity: Informational

The `getLockUntilTimestampLookup` and `setAddressPermissions` functions in the `RestrictedToken` contract can be made external for gas savings.

Recommendation

We recommend making the `getLockUntilTimestampLookup` and `setAddressPermissions` functions external.

11. Redundant value check in burn function

Severity: Informational

The `require` check in `L451` of the `RestrictedToken`'s `burn` function is redundant as the check is performed in ERC-20's `_burn`.

Recommendation

We recommend removing the `require` check in the `RestrictedToken`'s `burn` function.

12. Values greater than 9 passed to `messageForTransferRestriction` return empty strings

Severity: Informational

`uint8`s passed to `TransferRules`' `messageForTransferRestriction` greater than 9 return an empty string. This can lead to hard to understand EVM error messages.

Recommendation

We recommend adding a `require` check to ensure uint8s passed to `messageForTransferRestriction` are less than 9.

13. Redundant if statement in `messageForTransferRestriction`

Severity: Informational

The if statement in `TransferRulesUpgrade`'s `messageForTransferRestriction` function is always true.

Recommendation

We recommend removing the if statement in `TransferRulesUpgrade`'s `messageForTransferRestriction` function.

14. Re-entrancy allows for arbitrary `_swap` writes

Severity: Informational

A nefarious `quoteToken` can be passed to `configureBuy` of `RestrictedSwap`, reentering `configureSwap`, and filling up the swap queue with multiple swaps for 1 call to `configureBuy`. This issue is classified as informational because no real security risk seems to exist.

Recommendation

We recommend adding a reentrancy guard to functions with external calls to untrusted ERC-20 tokens.