SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Citizend_Ltd
Date: April 25th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Citizend_Ltd.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token; Token sale; Token vesting
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://website.com
Timeline	05.04.2022 - 25.04.2022
Changelog	14.04.2022 - Initial Review 25.04.2022 - Revise

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Citizend_Ltd (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

```
Repository:
      https://github.com/subvisual/discoveryDAO
Commit:
      587999addcc3feec4077c7209041926f9345fdfe
Documentation: Yes
JS tests: Yes
Contracts:
      packages/contracts/contracts/token/Vesting.sol
      packages/contracts/contracts/token/IVesting.sol
      packages/contracts/contracts/token/Sale.sol
      packages/contracts/contracts/token/ISale.sol
      packages/contracts/contracts/token/Citizend.sol
      packages/contracts/contracts/test/MockSale.sol
      packages/contracts/contracts/test/MockERC20.sol
      packages/contracts/contracts/RisingTide/TestRisingTideWithStaticAmoun
ts.sol
      packages/contracts/contracts/RisingTide/TestRisingTideWithCustomAmoun
ts.sol
      packages/contracts/contracts/RisingTide/RisingTide.sol
      packages/contracts/contracts/libraries/Math.sol
      packages/contracts/contracts/libraries/DateTime.sol
      packages/contracts/contracts/fractal_registry/FractalRegistry.sol
```

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided superficial functional requirements and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. The code follows style guide recommendations. Unit tests are provided. Code is commented. NatSpecs are present.

Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clear and transparent.

Security score

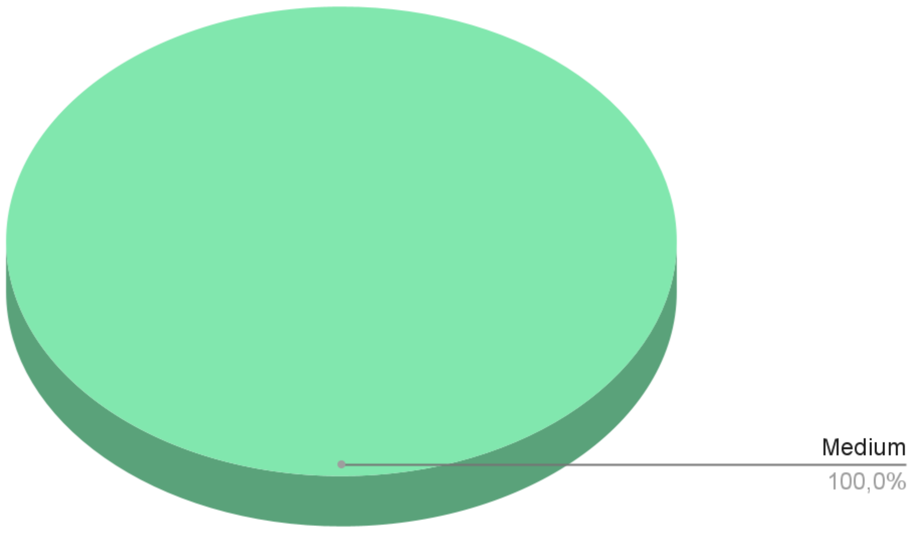
As a result of the audit, security engineers found **1** medium severity issue. The security score is **10** out of **10**. All found issues are displayed in the “Issues overview” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **6.5**

1
2
3
4
5
6
7
8
9
10
You are here

Graph 1. The distribution of vulnerabilities after the audit.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. **No tokens for refunds.**

As the documentation states: `Privileged account(s) can withdraw $aUSD after the sale is over (except money meant for refunds)`. The underlined part is not fulfilled with the smart contract.

Right now, the `withdraw` function returns all amount of `paymentToken` left on the contract after the sale ends.

Contracts: Sale.sol

Function: withdraw

Recommendation: subtract the amount needed for refunds from the token balance before withdrawing.

Status: Fixed (Revised Commit: 587999a)

■■ Medium

1. **Documentation inconsistency.**

The provided documentation states: `Total supply: 100 Million $CTND` while in the contract's code, we can see that the supply minted to the contract creator is: `1e9 ether`, which is 1 Billion.

Contracts: Citizend.sol

Function: constructor

Recommendation: correct either documentation or the contract code.

Status: Fixed (Revised Commit: 587999a)

■ Low

1. **Reading the state in the loop.**

Accessing the `sales.length` in the condition statement of the for-loop will make it call the state in each iteration and burn excess gas.

Contracts: Vesting.sol

Function: refund, totalAllocatedPublic

Recommendation: read the length into the local memory variable and then use it in the condition statement.

Status: Fixed (Revised Commit: 587999a)

2. **Unused import statement.**

Contracts import “hardhat/console.sol” while not using its functionality.

Contracts: RisingTide.sol, Sale.sol, Vesting.sol

Recommendation: remove unused import statement.

Status: Fixed (Revised Commit: 587999a)

3. **A public function that could be declared external.**

Public functions that are never called by the contract should be declared **external**.

Contracts: Citizend.sol, RisingTide.sol, FractalRegistry.sol, Vesting.sol

Function: Citizend.pause, Citizend.unpause, RisingTide.risingTide_validating, FractalRegistry.getFractalId, FractalRegistry.addUserAddress, FractalRegistry.isUserInList, FractalRegistry.addUserToList, FractalRegistry.removeUserFromList, FractalRegistry.addDelegate, FractalRegistry.removeDelegate, Vesting.totalAllocated,

Recommendation: use the **external** attribute for functions never called from the contract.

Status: Fixed (Revised Commit: 587999a)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.