HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Codex
Date: May 9th, 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 Codex.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Timeline	21.03.2022 - 06.05.2022
Changelog	25.03.2022 - Initial Review 14.04.2022 - Revision 27.04.2022 - Revision 09.05.2022 - Revision

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

Table of contents

Introduction 4

Scope 4

Executive Summary 5

Severity Definitions 7

Findings 8

Recommendations 12

Disclaimers 13

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

Introduction

Hacken OÜ (Consultant) was contracted by Codex (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/CDEXonAltHash/Codex-Rewards-Platform
Commit:
d364d0ef9258dd468f8202a352c58724d6b65638
6ec987cf357d337a042e3d4c209f37466f5db220 (revision)
395a888b54cc9fabbaed92bf068ff93ef4f3c433 (revision)
76d404b94ffbfc87b9e7c25633f8de580366778e (revision)
Technical Documentation: Yes
(https://github.com/CDEXonAltHash/Codex-Rewards-Platform/blob/main/readme/cdex_rewards.sol.md)
JS tests: No
Contract: CDEX_rewards.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

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

Executive Summary

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

Documentation quality

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

Code quality

The total CodeQuality score is **5** out of **10**. No unit tests were provided.

Architecture quality

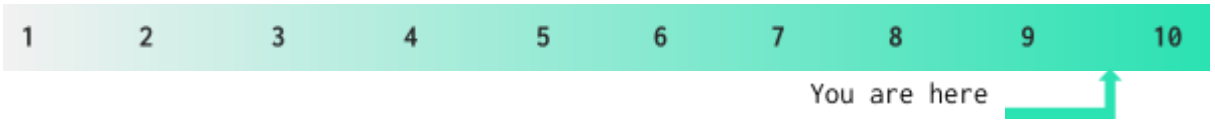
The architecture quality score is **10** out of **10**. All the logic is clear.

Security score

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

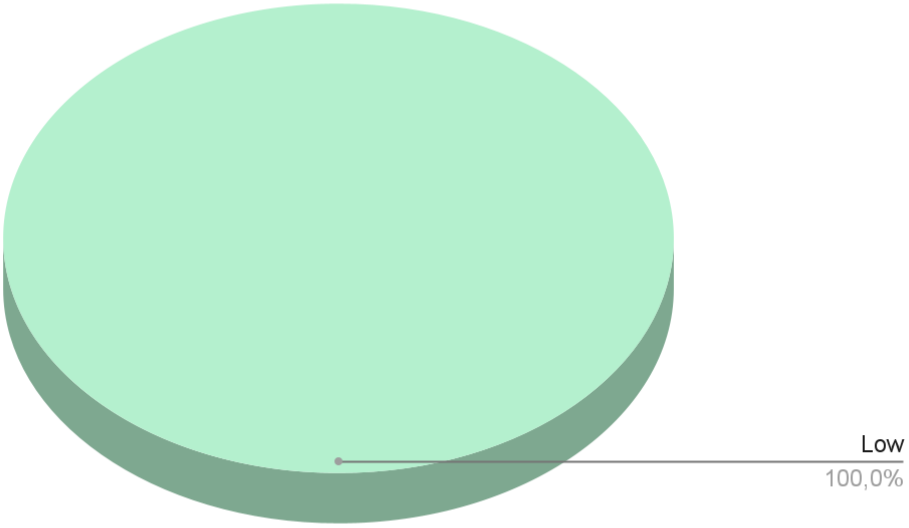
Summary

According to the assessment, the Customer's smart contract has the following score: **9.5**



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

Graph 1. The distribution of vulnerabilities after the audit.



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

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

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

Findings

Critical

1. **Depositing logic is corrupted.**

   On *depositTokens* *depositedRewardTokens* is increasing with a full amount of tokens (including reserved for loyalty bonuses), but it is not decreasing on transferring loyalty bonuses. In such a way, it could lead to double-spending.

   Contract: CDEX_rewards.sol

   Functions: depositTokens

   Recommendation: review and fix this logic.

   Status: Fixed (Revised Commit: 6ec987c)

2. **Depositing logic is corrupted.**

   depositedRewardTokens consists of 3 parts: pending rewards, reserved for paying rewards in future, and not involved tokens. When setting new reward rates in *notifyRewardAmount*, the code should use only reserved and not involved parts, pending rewards must not be touched.

   Example: user stakes some assets, tokens are deposited and rewards are notified, period is finished, rewards could be notified again despite them should be paid to the user on *getReward*.

   In such a way, setting rates during the staking period may lead to double spending of pending rewards.

   Contract: CDEX_rewards.sol

   Functions: notifyRewardAmount

   Recommendation: you should separate rewards that users may get for a passed period and rewards that users are not given yet (rewards that could be included in new reward rate calculation).

   Status: Fixed (Revised Commit: 76d404b)

High

1. **Users can get bonuses even if they should not.**

   The user can stake a small amount of money, but before claiming rewards add a big amount to get the bonus and then take it back. This can be easily exploited by using a flash loan service.

   Contract: CDEX_rewards.sol

   Function: getReward

   Recommendation: add the bonus to the reward in *updateReward* modifier.

   Status: Fixed (Revised Commit: 6ec987c)

2. **Unsafe functions.**

If these functions are called when there are any pending rewards, the logic may be corrupted: current token will be blocked on the contract; and there would be no new tokens, but the deposited value will not be zero.

Contract: CDEX_rewards.sol

Functions: setTokenContract, setRankingContract

Recommendation: implement checks to have no pending rewards, force unstaking function may be implemented for this purpose.

Status: Fixed (Revised Commit: 6ec987c)

3. **Unsafe functions.**

   If these functions are called when there are any pending stakes, the logic may be corrupted: current token will be blocked on the contract; and there would be no new tokens, but the deposited value will not be zero; removing users from new empty ranking service may fail too.

   According to the comment the functions are called only once on contract creation, so they could be declared as *internal*.

   Contract: CDEX_rewards.sol

   Functions: setTokenContract, setRankingContract

   Recommendation: make the functions *internal*.

   Status: Fixed (Revised Commit: 76d404b)

■■ Medium

1. **Check if *transfer* and *transferFrom* have finished successfully**

   Transfer can fail without reverting transaction according to internal reasons.

   Contract: CDEX_rewards.sol

   Functions: stake, withdraw, getReward, depositTokens

   Recommendation: check the return value of the functions and revert the transaction if it is false.

   Status: Fixed (Revised Commit: 6ec987c)

2. **Violation of ERC-20 standard.**

   CDEXTokenContract looks like an ERC-20 contract, but it does not correspond fully.

   Contract: CDEX_rewards.sol

   Interface: CDEXTokenContract

   Recommendation: *transfer* method should return a boolean value.

Status: Fixed (Revised Commit: 6ec987c)

3. **Key-value storage is wrongly defined.**

   In terms of the provided definition, balance is key, and user address is value, but there could be several users with equal balance.

   Contract: CDEX_rewards.sol

   Interface: CDEXRankingContract

   Recommendation: rename address as key and balance as value, check the storage implementation.

   Status: Fixed (Revised Commit: 6ec987c)

4. **Reserving too much money for bonuses.**

   loyaltyBonusTotal is set as the sum of loyalty bonuses, but only one of them is applied, so it is enough to set the maximum of them.

   Contract: CDEX_rewards.sol

   Function: setLoyaltyTiersBonus

   Recommendation: calculate *loyaltyBonusTotal* as the maximum of loyalty bonuses.

   Status: Fixed (Revised Commit: 6ec987c)

5. **Wrong ordered loyalty bonuses.**

   The functions do not check whether parameters are given descending order. If they would be provided in ascending order the logic of loyalty bonuses will be corrupted.

   Contract: CDEX_rewards.sol

   Functions: setLoyaltyTiersBonus, setLoyaltyTiers

   Recommendation: implement checks.

   Status: Fixed (Revised Commit: 6ec987c)

6. **Missing validation of transfer results.**

   Results of the “transfer” function are ignored.

   If the underlying token returns a value instead of throwing, the contract logic will be broken.

   Contract: CDEX_rewards.sol

   Functions: stake, withdraw

   Recommendation: implement corresponding validations.

   Status: new

Low

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

1. **Floating pragma**

   The contracts use floating pragma ^0.4.21.

   Contract: CDEX_rewards.sol

   Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

   Status: Fixed (Revised Commit: 6ec987c)

2. **Unused event**

   Event *Recovered* is defined but never used.

   Contract: CDEX_rewards.sol

   Recommendation: remove this event.

   Status: Fixed (Revised Commit: 6ec987c)

3. **Functions that can be declared as external**

   In order to save Gas, public functions that are never called in the contract should be declared as *external*.

   Contract: CDEX_rewards.sol

   Function: setTokenContract, setRankingContract, depositTokens, notifyRewardAmount

   Recommendation: aforementioned should be declared as *external*.

   Status: Fixed (Revised Commit: 6ec987c)

4. **Non-essential code**

   In order to save Gas, code that could be skipped should be skipped.

   committedRewardTokens duplicate *rewards* logic and keeps sum of all actual rewards, so it could be removed.

   reservedRewardTokens is not essential as it stores the amount of available money for paying rewards that is carefully controlled by *rewardRate*.

   depositedLoyaltyBonus stores non-essential information if least of the tokens allocated for it are not reused.

   Contract: CDEX_rewards.sol

   Function: getReward, depositTokens, notifyRewardAmount, updateReward

   Recommendation: remove using of the variables that are not essential.

   Status: New

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

Recommendations

1. Your contract reserves the maximum possible amount for loyalty bonuses, but not all of them are spent. In such a way some tokens are locked on the contract. You may carefully reuse them in the *notifyRewardAmount* function.

   Contracts: CDEX_rewards.sol

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

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.