SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hedgey
Date: March 31st, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Hedgey.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC721 token; OTC Exchange
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.hedgey.finance/
Timeline	15.03.2022 - 31.03.2022
Changelog	22.03.2022 - Initial Review 31.03.2022 - Revise

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Hedgey (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/hedgey-finance/NFT_OTC_Core
Commit:
b43c40d0a44e72d96a2d68dec1bcab7748571807
Technical Documentation: Yes (README.md)
JS tests: Yes (test)
Contracts:
libraries/TransferHelper.sol
libraries/NFTHelper.sol
HedgeyOTC.sol
CeloHedgeyOTC.sol
FuturesNFT.sol
CeloFuturesNFT.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 NatSpec, functional and technical requirements. Total Documentation Quality score is **8** out of **10**.

Code quality

The total CodeQuality score is **8** out of **10**. Unit tests were provided. Good code comments. Not following style guidelines for line length.

Architecture quality

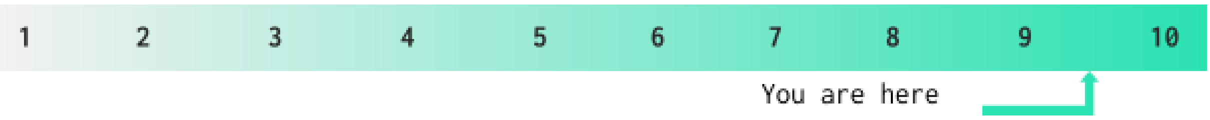
The architecture quality score is **8** out of **10**. All the logic is implemented in separate files. Pretty good architecture but have room for improvement.

Security score

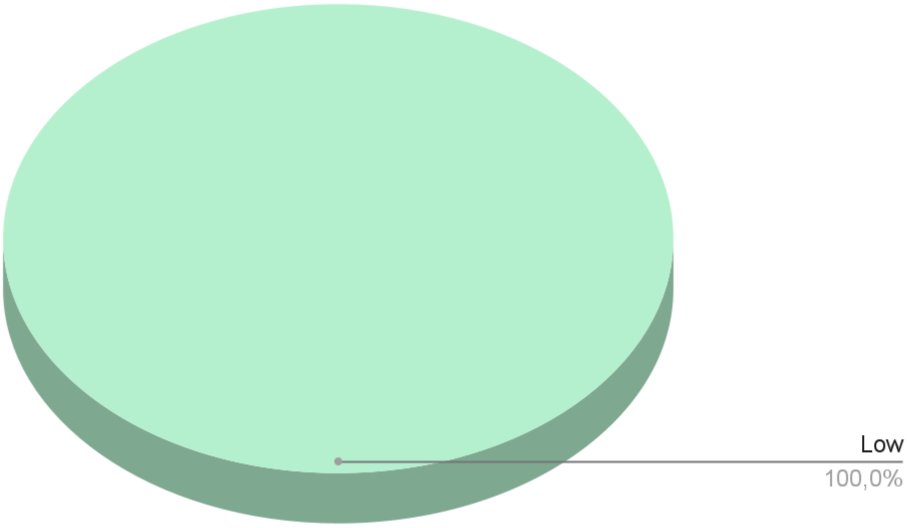
As a result of the audit, security engineers found **1** medium and **9** low severity issues. The security score is **10** out of **10**. All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.4**



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

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

Using “transfer()” function.

Starting EIP 1884 and Istanbul hard fork, it is not recommended to use either “transfer()” or the “send()” functions to send ether to the address. Many details can be found here.

Contracts: FuturesNFT.sol, HedgeyOTC.sol

Function: _withdraw

Recommendation: Use the ‘**to.transfer(_amt)**’ construction instead.

Status: Fixed (Revised Commit: 06a3c29c3b)

🟦 Low

1. **Using storage instead of memory.**

   Using the local storage variable will not allocate memory for its value but instead will do calls to the storage each time accessing it.

   Contracts: FuturesNFT.sol, CeloFuturesNFT.sol

   Function: _redeemNFT

   Recommendation: use **memory** for a local variable to save gas.

   Status: Fixed (Revised Commit: 06a3c29c3b)

2. **Using storage instead of memory.**

   Using the local storage variable will not allocate memory for its value but instead will do calls to the storage each time accessing it.

   Contracts: CeloHedgeyOTC.sol, HedgeyOTC.sol

   Function: buy

   Recommendation: use **memory** for a local variable to save gas and then change updated values to the state at the end of the function.

   Status: Fixed (Revised Commit: 06a3c29c3b)

3. **Outdated version of solidity.**

   While the latest solidity version is 0.8.13, contracts are still written using the half-year-old 0.8.7 compiler version.

Contracts: CeloFuturesNFT.sol, CeloHedgeyOTC.sol, FuturesNFT.sol, HedgeyOTC.sol

Recommendation: use a more recent compiler version.

Status: Fixed (Revised Commit: 06a3c29c3b)

4. **Benign reentrancy.**

   The function contains reentrancy. The reentrancy is benign because its exploitation would have the same effect as two consecutive calls.

   Contract: CeloHedgeyOTC.sol

   Function: create

   Recommendation: place `SafeERC20.safeTransferFrom` call after the state changes (deals[d ++]).

   Status: Fixed (Revised Commit: 06a3c29c3b)

5. **Benign reentrancy.**

   The function contains reentrancy. The reentrancy is benign because its exploitation would have the same effect as two consecutive calls.

   Contract: HedgeyOTC.sol

   Function: create

   Recommendation: place token transfer calls after the state changes (deals[d ++]).

   Status: Fixed (Revised Commit: 06a3c29c3b)

6. **Benign reentrancy.**

   The function contains reentrancy. The reentrancy is benign because its exploitation would have the same effect as two consecutive calls.

   Contract: CeloHedgeys.sol

   Function: createNFT

   Recommendation: place `_safeMint` and `SafeERC20.safeTransferFrom` calls after the state changes (futures[newItemId]).

   Status: Fixed (Revised Commit: 06a3c29c3b)

7. **Benign reentrancy.**

   The function contains reentrancy. The reentrancy is benign because its exploitation would have the same effect as two consecutive calls.

   Contract: FuturesNFT.sol

   Function: createNFT

   Recommendation: place `_safeMint` and `SafeERC20.safeTransferFrom` calls after the state changes (futures[newItemId]).

Status: Fixed (Revised Commit: 06a3c29c3b)

8. **No event emitting.**

   Changing the crucial state value requires emitting an event.

   Contracts: CeloFuturesNFT.sol, FuturesNFT.sol

   Function: updateBaseURI

   Recommendation: emit an event on baseURI change.

   Status: Fixed (Revised Commit: 06a3c29c3b)

9. **Uninformative function parameter.**

   Function parameter `_d` neither self-descriptive nor has a NatSpec description.

   Contracts: CeloHedgeyOTC.sol, HedgeyOTC.sol

   Functions: close, buy

   Recommendation: rename the variable to something more informative, such as a `dealId`.

   Status: Not fixed

Disclaimers

Hacken Disclaimer

The smart contracts were 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.