

**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** Hedgey

**Date:** August 09<sup>th</sup>, 2022

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for Hedgey
<b>Approved By</b>	Evgeniy Bezuglyi   SC Audits Department Head at Hacken OU Noah Jelich   Senior Solidity SC Auditor at Hacken OU
<b>Type</b>	ERC721 token
<b>Platform</b>	EVM
<b>Network</b>	Ethereum
<b>Language</b>	Solidity
<b>Methods</b>	Manual Review, Automated Review, Architecture Review
<b>Website</b>	<a href="https://hedgey.finance/">https://hedgey.finance/</a>
<b>Timeline</b>	21.07.2022 - 08.08.2022
<b>Changelog</b>	22.07.2022 - Initial Review 02.08.2022 - Second Review 08.08.2022 - Third Review



## Table of contents

Introduction	4
Scope	4
Severity Definitions	6
Executive Summary	7
Checked Items	8
System Overview	11
Findings	12
Disclaimers	14

## 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:

### Initial review scope

**Repository:**

[https://github.com/hedgey-finance/NFT\\_OTC\\_Core/](https://github.com/hedgey-finance/NFT_OTC_Core/)

**Commit:**

adba240e6888adfd2358116de11e6215c9be7938

**Technical Documentation:**

Type: Technical description

**Integration and Unit Tests:** Yes

**Contracts:**

File: ./contracts/NonTransferrableNFT.sol

SHA3: 249ec8278c12f39c75cef7d34d6f0305bd90a5ef839d039adff50fd5b80b6760

File: ./contracts/libraries/TransferHelper.sol

SHA3: b50cfa9a8492c9888c47612e8350b9402b5cdd7cd5890eea13b855d14529409e

File: ./contracts/interfaces/IWETH.sol

SHA3: ec07545517d331eec6ccb3f7dc620b82b3fe877fec00a849ada1c03338e7bf0f

### Second review scope

**Repository:**

[https://github.com/hedgey-finance/NFT\\_OTC\\_Core/](https://github.com/hedgey-finance/NFT_OTC_Core/)

**Commit:**

e6c8a05601e2546e3d0d776ce3f8b7fa5968aa37

**Technical Documentation:**

Type: Technical description

**Integration and Unit Tests:** Yes

**Contracts:**

File: ./contracts/interfaces/IWETH.sol

SHA3: 702a6c595a3ea5c615f3aff3008b760d6d70f4c4e23f06f3ea4329160d019ee2

File: ./contracts/libraries/TransferHelper.sol

SHA3: 6afcf1fb04f38c7900ddb41c3e8095d0dd84fb481665aaeda3a6879aba4a3f84

File: ./contracts/NonTransferrableNFT.sol

SHA3: 5c8cf96d09a5c1285b0ba1c0b3e3e2ed8fdc8bd8419ba724d767c4bb9fb9a3c54

### Third review scope

**Repository:**

[https://github.com/hedgey-finance/NFT\\_OTC\\_Core/](https://github.com/hedgey-finance/NFT_OTC_Core/)

**Commit:**

67055a7d4f7517f15c5c85bed67fde0982eb4c2c

**Technical Documentation:**

Type: Technical description

**Integration and Unit Tests:** Yes

**Contracts:**



File: ./contracts/interfaces/IWETH.sol  
SHA3: 702a6c595a3ea5c615f3aff3008b760d6d70f4c4e23f06f3ea4329160d019ee2

File: ./contracts/libraries/TransferHelper.sol  
SHA3: 6afcf1fb04f38c7900ddb41c3e8095d0dd84fb481665aaeda3a6879aba4a3f84

File: ./contracts/NonTransferrableNFT.sol  
SHA3: 8da626324831ec64d48170a76befba1b297b93b3ca68cb61659c585b9384771c

## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
<b>High</b>	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.
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
<b>Low</b>	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

## Executive Summary

The score measurement details can be found in the corresponding section of the [methodology](#).

### Documentation quality

The total Documentation Quality score is **5** out of **10**. A technical description is provided as the detailed comments in the code. Functional requirements are not provided.

### Code quality

The total CodeQuality score is **10** out of **10**. Most of the code follows official language style guides. Unit tests were provided, some out of scope tests are failing.

### Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clear. The development environment was provided.

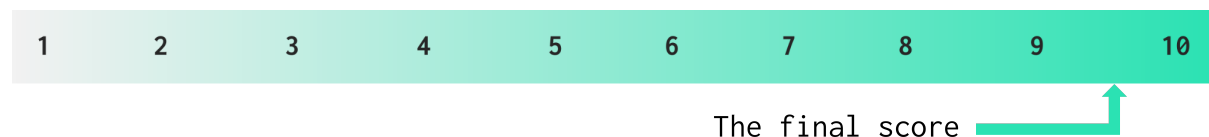
### Security score

As a result of the audit, the code contains **no** 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: **9.5**.



## Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	<a href="#">SWC-100</a> <a href="#">SWC-108</a>	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	<a href="#">SWC-101</a>	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	<a href="#">SWC-102</a>	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	<a href="#">SWC-103</a>	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	<a href="#">SWC-104</a>	The return value of a message call should be checked.	Passed
Access Control & Authorization	<a href="#">CWE-284</a>	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	<a href="#">SWC-106</a>	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	<a href="#">SWC-107</a>	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	<a href="#">SWC-110</a>	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	<a href="#">SWC-111</a>	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	<a href="#">SWC-112</a>	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	<a href="#">SWC-113</a> <a href="#">SWC-128</a>	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	<a href="#">SWC-114</a>	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	<a href="#">SWC-115</a>	tx.origin should not be used for	Passed



through tx.origin		authorization.	
Block values as a proxy for time	<a href="#">SWC-116</a>	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	<a href="#">SWC-117</a> <a href="#">SWC-121</a> <a href="#">SWC-122</a> <a href="#">EIP-155</a>	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	<a href="#">SWC-119</a>	State variables should not be shadowed.	Passed
Weak Sources of Randomness	<a href="#">SWC-120</a>	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	<a href="#">SWC-125</a>	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
Calls Only to Trusted Addresses	<a href="#">EEA-Lev e1-2</a> <a href="#">SWC-126</a>	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	<a href="#">SWC-131</a>	The code should not contain unused variables if this is not <a href="#">justified</a> by design.	Passed
EIP standards violation	<a href="#">EIP</a>	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
<b>Style guide violation</b>	<b>Custom</b>	Style guides and best practices should be followed.	Passed
<b>Requirements Compliance</b>	<b>Custom</b>	The code should be compliant with the requirements provided by the Customer.	Passed
<b>Environment Consistency</b>	<b>Custom</b>	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
<b>Secure Oracles Usage</b>	<b>Custom</b>	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
<b>Tests Coverage</b>	<b>Custom</b>	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
<b>Stable Imports</b>	<b>Custom</b>	The code should not reference draft contracts, that may be changed in the future.	Passed

## System Overview

*NonTransferrableNFTs* is an ERC-721 project with the following contracts:

- *NonTransferrableNFTs* – is a non-transferrable ERC-721 token.

It has the following attributes:

- Name: NonTransferrable Hedgeys
- Symbol: NTHG

The contract allows to lock any tokens in any amount ( $> 0$ ) for any amount of time. When locking, the NFT is minted to the defined by the user holder address. When lock time expires, it is possible to redeem the tokens: the NFT is burnt, and the locked tokens are transferred to the holder address.

- *TransferHelper* – is a library that helps to transfer tokens and coins, used in *NonTransferrableNFTs* contract.
- *IWETH* – is an interface for handling ETH wrapping into WETH and unwrapping WETH into ETHm used in *TransferHelper* library.

## Risks

- *NonTransferrableNFTs* token can not be transferred.

## Findings

### ■■■■ Critical

No critical severity issues were found.

### ■■■ High

#### 1. Funds lock

The contract accepts the native coins (`receive` function), but there is no possibility to withdraw them.

Funds can be locked in the contract.

**File:** ./contracts/NonTransferrableNFT.sol

**Contract:** NonTransferrableNFT

**Function:** receive

**Recommendation:** Add the function for native coins withdrawal.

**Status:** Fixed (Revised commit:  
e6c8a05601e2546e3d0d776ce3f8b7fa5968aa37)

#### 2. Access control violation

The `updateBaseURI` function is not protected, and is callable by anyone.

Therefore, any user can update the base URI value (once).

**File:** ./contracts/NonTransferrableNFT.sol

**Contract:** NonTransferrableNFT

**Function:** updateBaseURI

**Recommendation:** Ensure that the `updateBaseURI` function is callable only by the admin user.

**Status:** Fixed (Revised commit:  
e6c8a05601e2546e3d0d776ce3f8b7fa5968aa37)

#### 3. Requirements violation

According to the comment in code, the locked WETH should be unwrapped and delivered in ETH during the redemption. (*“handles weth in case WETH is being held - this allows us to unwrap and deliver ETH upon redemption of a timelocked NFT with ETH”*)

However, the WETH is not unwrapped and is delivered in WETH during the redemption.

**File:** ./contracts/NonTransferrableNFT.sol

**Contract:** NonTransferrableNFT

**Function:** `_redeemNFT`

**Recommendation:** Implement the `code` according to requirements.

**Status:** Fixed (Revised `commit: 67055a7d4f7517f15c5c85bed67fde0982eb4c2c`)

## ■ ■ Medium

### Failing tests

The `Cannot Transfer: wallet cannot transfer a non-transferrable NFT` test is failing.

**Recommendation:** Ensure that all the test cases are passing.

**Status:** Fixed (Revised `commit: e6c8a05601e2546e3d0d776ce3f8b7fa5968aa37`). Out of scope: 19 other project tests fail; it is recommended to ensure that all the project tests are passing.

## ■ Low

### 1. Block values as a proxy for time using

The contract uses `block.timestamp` for time calculations. It is not precise and safe.

**File:** `./contracts/NonTransferrableNFT.sol`

**Contract:** `NonTransferrableNFT`

**Functions:** `createNFT, _redeemNFT`

**Recommendation:** It is recommended to avoid using `block.timestamp` in the time calculations. Alternatively, it is safe to use oracles.

**Status:** Mitigated. The Customer comment: “per the blogs block time will be consistent at 12 seconds post merge, and since the usage for us is only that events can happen after a certain `block.timestamp`, we feel this is as good or better than a centralized oracle like chainlink (and significantly less expensive)”.

### 2. Floating pragma

Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs.

**Files:** `./contracts/libraries/TransferHelper.sol, ./contracts/interfaces/IWETH.sol`

**Recommendation:** Consider locking the pragma version whenever possible.

**Status:** Fixed (Revised `commit: e6c8a05601e2546e3d0d776ce3f8b7fa5968aa37`)

## 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

### 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, Consultant cannot guarantee the explicit security of the audited smart contracts.