HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: iRocket
**Date**: August 03th, 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 iRocket |
| **Approved By** | Evgeniy Bezuglyi \| SC Audits Department Head at Hacken OU<br>Noah Jelich \| Senior Solidity SC Auditor at Hacken OU |
| **Type** | ERC-721 token |
| **Platform** | EVM |
| **Network** | Ethereum |
| **Language** | Solidity |
| **Methods** | Manual Review, Automated Review, Architecture Review |
| **Website** | https://irocket.io/ |
| **Timeline** | 18.07.2022 - 03.08.2022 |
| **Changelog** | 21.07.2022 - Initial Review<br>26.07.2022 - Second Review<br>03.08.2022 - Third Review |

## Table of contents

## Introduction

Hacken OÜ (Consultant) was contracted by iRocket (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/iRocket-io/NEOGEN-contract
**Commit:**
8a2b147806ba7952c046eca104d4882db71dfef2
**Technical Documentation:**
Type: Technical description
**Integration and Unit Tests:** Yes
**Contracts:**
File: ./contracts/libs/Base64.sol
SHA3: 130559fbc14d857d79ed19c7f49f865503719a3b5d13126fc15d5e382890992d

File: ./contracts/libs/ERC2981.sol
SHA3: 6ed00f2f2c4fa826b60df515abc58301045b4ded8fa646ef497408ab769bbbbf

File: ./contracts/NFTCollection.sol
SHA3: 9a7530776ede259cddeff3770b3f163dd7852b7e82d62ce5834428e4a1dd0bf0

File: ./contracts/NFTCollectionContract.sol
SHA3: 0de4a94f89c4e2bc5586be6a2a2190777b614db8435562cfc0b7d5f0dec4e011

### Second review scope

**Repository:**
https://github.com/iRocket-io/NEOGEN-contract
**Commit:**
05735253cff5ebac3659f79bcc15c1ddbd2a6561
**Technical Documentation:**
Type: Technical description
**Integration and Unit Tests:** Yes
**Contracts:**
File: ./contracts/NFTCollection.sol
SHA3: c2d6619fd236950c3bc1b7fce648931e1bb214f1cdf2faf2c73e943f560a5cd8

File: ./contracts/NFTCollectionContract.sol
SHA3: 3f6190c0b39d6b7d2865b626d399f907a3d27f20f57187612093fc38925f3ca9

### Third review scope

**Repository:**
https://github.com/iRocket-io/NEOGEN-contract/tree/test
**Commit:**
d061fe10f7bbad9c0652e05b9afc3998590f959d
**Technical Documentation:**
Type: Technical description
**Integration and Unit Tests:** Yes
**Contracts:**

```
      File: ./contracts/NEOGEN.sol
      SHA3: 1f41f20fad1e948589a9372f0733821090e890acb7610249936c9487af5a61a3

      File: ./contracts/NEOGENContract.sol
      SHA3: ad201bca81c36d082be81a01019041a5ede57cdf4c42c3d894402a73f0422a67
```

## 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. |

## Executive Summary

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

### Documentation quality

The total Documentation Quality score is **6** out of **10**. Functional requirements are not comprehensive. A technical description is provided as comments in the code.

### Code quality

The total CodeQuality score is **9** out of **10**. Most of the code follows official language style guides. The unit tests are provided, but some do not pass (Events tests).

### Architecture quality

The architecture quality score is **8** out of **10**. The architecture is generally clear. However, an unnecessary inheritance pattern is used (check Low 1). The configured development environment was provided.

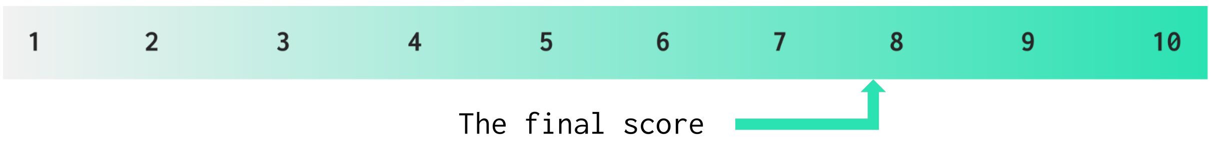
### Security score

As a result of the audit, the code contains **2** medium, and **2** low severity issues. The security score is **8** 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: **7.9**.



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

| | | | |
|---|---|---|---|
| **through tx.origin** | | authorization. | |
| **Block values as a proxy for time** | SWC-116 | Block numbers should not be used for time calculations. | Failed |
| **Signature Unique Id** | SWC-117<br>SWC-121<br>SWC-122<br>EIP-155 | 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** | SWC-119 | State variables should not be shadowed. | Passed |
| **Weak Sources of Randomness** | SWC-120 | Random values should never be generated from Chain Attributes or be predictable. | Not Relevant |
| **Incorrect Inheritance Order** | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. | Passed |
| **Calls Only to Trusted Addresses** | EEA-Level-2<br>SWC-126 | All external calls should be performed only to trusted addresses. | Not Relevant |
| **Presence of unused variables** | SWC-131 | The code should not contain unused variables if this is not justified by design. | Passed |
| **EIP standards violation** | EIP | 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. | |
| **Style guide violation** | Custom | Style guides and best practices should be followed. | Passed |
| **Requirements Compliance** | Custom | The code should be compliant with the requirements provided by the Customer. | Not Relevant |
| **Environment Consistency** | Custom | The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | Passed |
| **Secure Oracles Usage** | Custom | 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 |
| **Tests Coverage** | Custom | 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. | Failed |
| **Stable Imports** | Custom | The code should not reference draft contracts, that may be changed in the future. | Passed |

## System Overview

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

- *NEOGENContract (NFTCollection* in 1st, 2nd reviews*)* – is an ERC-721 contract. Name, symbol, and maximal supply are defined during the contract deployment. The tokens are minted in the process of tokens selling. There are four phases of sale:
  - Presale: Users should be whitelisted and provide the proof (Merkle Tree) to participate in the presale. Each user can buy tokens only once per the presale phase. It is allowed to buy up to 3 tokens per one transaction.
  - Public sale: Any users can participate. It is allowed to buy up to 3 tokens per one transaction.
  - Free sale: Users should be freelisted and provide the proof (Merkle Tree) to mint tokens for free. Each user can mint one token per free sale. It is allowed to buy 1 token per one transaction.

    The Presale and Free sale should start and not be finished to be available. The Public sale should start to be available. Start time for all phases and finish time for presale, free sale are defined when deployment and can be changed.

    The token price for Presale and Public sale phases (are defined when the contract deployment, the price can be changed). Users should pay the fee in ETH.
  - INO sale: Is performed by the “ADMIN” user. It is allowed to mint up to 5000 tokens per one transaction.

  The paid fees are transferred to the treasury address by the “ADMIN” user.
  The contract supports ERC-2981 (royalty amount and receiver information defining).
- *NEOGEN (NFTCollectionContract* in 1st, 2nd reviews*)* – is the contract that implements *NFTCollection* contract.

### Privileged roles

- The “ADMIN_ROLE” of *NFTCollection* contract allows to transfer “ADMIN_ROLE” to another address, mint tokens in INO phase, update runtime configuration (tokens mint price, tokens base and pre-reveal URI, Presale and Public sale start times, Presale end time, Freemint phase start and end time, whitelist and freelist Merkle roots, royalties receiver and basis points, treasury address), withdraw paid fees to the treasury address.
- The “DEFAULT_ADMIN_ROLE” of NFTCollection contract allows to transfer contract ownership and manage other roles.

## Risks

- *NFTCollection* contract can not be instantiated as it is not initializable. It should be inherited with setting *false* to the *_preventInitialization* variable.

## Findings

### ■■■■ Critical

1. **Data consistency**

The contract uses the *royaltyInfo* function from OpenZeppelin contract and does not override it. The royalty information is stored in the *_runtimeConfig.royaltiesBps*, *_runtimeConfig.royaltiesAddress*, *ROYALTIES_BASIS* variables in the *NFTCollection* contract; but the *royaltyInfo* function from OpenZeppelin contract returns the information from *_tokenRoyaltyInfo*, *_defaultRoyaltyInfo*, *_feeDenominator()* contract values.

Therefore, the *royaltyInfo* function will always return zero address as the royalty receiver and zero as a royalty amount. Due to this, the royalty will never be paid.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function**: royaltyInfo

**Recommendation**: Override the *royaltyInfo* function with returning appropriate royalty values in it.

**Status**: Fixed (Revised commit: d061fe10f7bbad9c0652e05b9afc3998590f959d)

### ■■■ High

1. **Invalid calculations**

Sufficient balance of reserved tokens is validated using the following condition: *amount <= addressList.length * reserveRemaining*.

This verification works incorrectly, as the fulfillment of this condition does not mean that the *reserveRemaining* is enough for minting the *amount* of tokens for the *addressList.length* addresses.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function**: freeMintByList

**Recommendation**: Replace *amount <= addressList.length * reserveRemaining* with the *addressList.length * amount <= reserveRemaining*.

**Status**: Functionality removed

2. **Data consistency**

There is no Presale finish time in the *_runtimeConfig* variable. It is not checked if the Presale phase finished in the *presaleActive* function.

Therefore, the Presale phase will always be available and occur simultaneously with Public sale.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function**: presaleActive

**Recommendation**: Define the Presale finish time in the *_runtimeConfig* variable and check if the Presale phase has not finished in the *presaleActive* function.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

3. **Requirements violation**

The *isFreelisted* function verifies if the user is freelisted. The free mint is executed by the user with "ADMIN_ROLE", and the users are not verified. The amount of tokens to be minted is passed via the functions (*reserveMint*, *freeMintByList*) parameters (*amount*).

Therefore, users who are not freelisted may get the tokens, and the freelisted users may not get the tokens or get them in the correct amount.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Functions**: isFreelisted, reserveMint, freeMintByList

**Recommendation**: Clarify the freeminting rules, ensure that users getting tokens are freelisted, and get tokens in the correct amounts.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

4. **Denial of Service vulnerability**

The *_transferOwnership* function revokes *ADMIN_ROLE* and *DEFAULT_ADMIN_ROLE* roles using the *_deploymentConfig.owner*, but the *_deploymentConfig.owner* may not be granted with these roles.

Therefore, *_revokeRole* function will revert, and the *_transferOwnership* function will be inoperable.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function**: _transferOwnership

**Recommendation**: Split the roles transferring into different functions or ensure that *_deploymentConfig.owner*, *ADMIN_ROLE* and *DEFAULT_ADMIN_ROLE* are the same.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

5. **Denial of Service vulnerability and requirements violation**

The *runtimeConfig* is not validated during the contract initialization. The *royaltiesBps* value can be bigger than the *ROYALTIES_BASIS* value.

Therefore, it may be impossible to sell tokens in the secondary marketplaces as the royalty will be calculated incorrectly, exceeding the token price.

The *baseURI* and *prerevealTokenURI* are not validated.

Due to this, the tokens` URIs may be empty.

*_runtimeConfig.royaltiesAddress* is not validated for non-zero value.

Therefore, the royalty tokens can be burnt. The secondary marketplaces may not validate the value, as the royalty receiver is usually validated in the well-known contracts (OpenZeppelin ERC-2981 implementation).

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Functions**: initialize, _validateRuntimeConfig

**Recommendation**: Validate the *runtimeConfig* on the contract initialization. Verify that *_runtimeConfig.royaltiesAddress* is non-zero.

**Status**: Fixed (Revised commit: d061fe10f7bbad9c0652e05b9afc3998590f959d)

6. **Access control violation**

During the contract initialization *msg.sender* is granted with the *ADMIN_ROLE* role, and this role is transferred to *deploymentConfig.owner* using the *_transferOwnership* function. The *_transferOwnership* function uses *_deploymentConfig.owner* for role revoking.

Therefore, if *msg.sender* and *deploymentConfig.owner* are different in the *initialize* function, the *ADMIN_ROLE* for *msg.sender* will not be revoked, and this user will have access to the crucial functions.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Functions**: isFreelisted, reserveMint, freeMintByList

**Recommendation**: Ensure that *msg.sender* and *deploymentConfig.owner* are equal in the *initialize* function.

**Status**: Fixed (Revised commit: d061fe10f7bbad9c0652e05b9afc3998590f959d)

## ■■ Medium

### 1. Using “greater than or equal to” operator for the payment check

When selling the tokens, it is checked if the payment sent (*msg.value*) is greater than or equal to the price.

Users can mistakenly send more coins, or the price can be changed, and users will pay more than needed.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function**: paymentProvided

**Recommendation**: Use “equal to” operator for the sent payment checking.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

### 2. Unsafe treasury address using

The functionality does not allow to change the *_deploymentConfig.treasuryAddress* value.

If the defined address is compromised, irrelevant, or wrongly specified, it will not be possible to change it.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Path**: _deploymentConfig.treasuryAddress

**Recommendation**: Allow to change the *_deploymentConfig.treasuryAddress* value.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

### 3. Missing events emit on changing important values

The contract does not emit any events after changing important values.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function:** updateConfig

**Recommendation**: Implement event emits after changing the contract values.

**Status**: Reported. The *UpdateRuntimeConfig* event does not log the updated configurations.

4. **Compilation issue**

The *NFTCollection* contract uses ^0.8.0 pragma, and overrides the *royaltyInfo* function without the *override* specifier.

Therefore, if the Solidity version lower than 0.8.8 is used, the contract will not compile.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Function**: royaltyInfo

**Recommendation**: Lock the pragma to the 0.8.8 version (or higher) or add the *override* keyword to the *royaltyInfo* function.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

5. **Failing tests**

The *Events* tests are failing.

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

**Status**: New

6. **Validations missing**

The *newOwner* parameter is not checked for the non-zero value in the *transferOwnership* function.

Therefore, the contract can lose ownership, and all the accessible for the owner functions will be inoperable.

The *_address* is not checked for non-equality with the *_deploymentConfig.owner* in the *revokeAdminRights* function.

If the owner revokes admin rights for their address, it would be impossible to transfer the ownership using the *transferOwnership* function.

**File:** ./contracts/NEOGENContract.sol

**Contract**: NEOGENContract

**Functions**: transferOwnership, revokeAdminRights

**Recommendation**: Validate the *newOwner* parameter for the non-zero value in the *transferOwnership* function; Ensure that the owner can not revoke the admin rights for their address in the *revokeAdminRights* function.

**Status**: New

## Low

### 1. The contract can be declared as abstract

*NFTCollection* contract is not initializable and should be inherited with setting *false* to the *_preventInitialization* variable.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Recommendation**: Declare the contracts as abstract.

**Status**: Reported

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

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

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Functions**: mintingActive, presaleActive

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

**Status**: Reported

### 3. Floating pragma

The project`s contracts use floating pragma ^0.8.0.

Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs. Contracts should be deployed with the same compiler version they have been tested thoroughly.

**Files:** ./contracts/NFTCollection.sol, ./contracts/libs/Base64.sol, ./contracts/libs/ERC2981.sol, ./contracts/NFTCollectionContract.sol

**Contracts:** NFTCollection, Base64, ERC2981, NFTCollectionContract

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

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

### 4. Modification of a well-known contract

Imported or copy-pasted contracts (like *SafeMath*, *Context*, *Ownable*, etc.) should not be modified to keep the code clear.

The Base64 is a modified version of the OpenZeppelin implementation of the contract of the same name.

**File:** ./contracts/libs/Base64.sol

**Contract**: Base64

**Recommendation**: Delete the modifications.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

5. **Functions that can be declared external**

There are public functions in the contracts that are not used internally.

“External” visibility uses less Gas.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Functions**: maxSupply, reservedSupply, tokensPerMint, treasuryAddress, mintPrice, publicMintStart, presaleMintStart, whitelistMerkleRoot, freeMintMerkleRoot, baseURI, metadataUpdatable, prerevealTokenURI, owner

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

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

6. **Default visibility usage**

There is a variable in the contract whose visibility is not defined explicitly.

This may lead to incorrect access to the variable.

**File:** ./contracts/NFTCollection.sol

**Contract**: NFTCollection

**Path:** ROYALTIES_BASIS

**Recommendation**: Define the variables' visibilities explicitly.

**Status**: Fixed (Revised commit: 05735253cff5ebac3659f79bcc15c1ddbd2a6561)

## 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.