HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Belong
Date: December 8, 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 Belong
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC721 system
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	
Timeline	14.10.2022 - 08.12.2022
Changelog	19.10.2022 - Initial Review 09.11.2022 - Second Review 22.11.2022 - Third Review 08.12.2022 - Fourth Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Belong (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://gitlab.com/nomadhub/smart-contracts
Commit:
64db0205730debaef6aa0b49befae348a7231a61
Documentation:
https://gitlab.com/nomadhub/smart-contracts/README.md
Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:

File: ./contracts/Factory.sol
SHA3: 009694f718bd7a253328a635d5ef545541f797525fc083e1a060e7a3119ab2bc

File: ./contracts/NFT.sol
SHA3: 76f326e6e0860c78fcf51ca0fefbf59dbf7e2aee4006e20b042b44e644dbfd36

File: ./contracts/ReceiverFactory.sol
SHA3: f10161f44fc3321486127f8ed3e3d00fa11918ad09bf842341798b1e6ac644b3

File: ./contracts/RoyaltiesReceiver.sol
SHA3: 5cbe96ad49437bff0d554091ebf41781fdcc32ea7a841eabeef42ae039cca51f

File: ./contracts/StorageContract.sol
SHA3: 0f2b77dd5ae73ac4009a36f733ae85e1f62ae5c9bb6d9faada45558cc7b29a0d

File: ./contracts/interfaces/IFactory.sol
SHA3: 9bdaf848c5279d39abd9dc9df2c8d5d2582008d724f7645dad87103f300dc3ff

File: ./contracts/interfaces/IStorageContract.sol
SHA3: ee451e995147b6271060d1eb6aef5282435d59cba18d13881f87486796646135

Second review scope

Repository:
https://gitlab.com/nomadhub/smart-contracts
Commit:
b8946732a842f85c71476242ff39a850c5615664
Documentation:
https://gitlab.com/nomadhub/smart-contracts/README.md
Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:

File: ./contracts/Factory.sol
SHA3: 750560d5dcf32b9a580e56e1f714bd4a487ba7b80a54087281a2fda2a2dd3898

File: ./contracts/NFT.sol
SHA3: e2698cc53d1dbf638530ce4512bc60c7e262519be134150b5f0e17d5535186e2

File: ./contracts/ReceiverFactory.sol
SHA3: 3f9dd893eb4619910733f4d50c6fafa2666edddb2f6d74b05ea5b90ea25ffdf1

File: ./contracts/RoyaltiesReceiver.sol
SHA3: d0424b8f878fcf49b9f570099be7d14ea785d9cd24eaa38c3a8b3f873c084782

File: ./contracts/StorageContract.sol
SHA3: 481cf386c217488088ecbf36108068311db90f4e1186733c6b062995393a0a1a

File: ./contracts/interfaces/IFactory.sol
SHA3: 29a722a89b5aad76b07771655415e8a640c077d9d56ca6a6f5a3678a43fdaea3

File: ./contracts/interfaces/IStorageContract.sol
SHA3: 6be4a89d242f8b161970ba538019be8e699251ba189232a10dd1a0d819e21cca

Third review scope

Repository:
https://gitlab.com/nomadhub/smart-contracts
Commit:
45f2029531e2965c648d8b425efd55c8c52b4cd3
Documentation:
https://gitlab.com/nomadhub/smart-contracts/README.md
Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:
File: ./contracts/Factory.sol
SHA3: 750560d5dcf32b9a580e56e1f714bd4a487ba7b80a54087281a2fda2a2dd3898

File: ./contracts/interfaces/IFactory.sol
SHA3: 30530929623b49eeaacf25917cba356caff3863bd295dae47d81acb2f8160ca3

File: ./contracts/interfaces/IStorageContract.sol
SHA3: 39634a25012b991dad7b952ad6802d577261a554850feb13bfed9798508783ab

File: ./contracts/NFT.sol
SHA3: 1f1805968510904a8d72e4db7fd63648aed01864ce81339d772a7d8bbb1b79fb

File: ./contracts/ReceiverFactory.sol
SHA3: 3f9dd893eb4619910733f4d50c6fafa2666edddb2f6d74b05ea5b90ea25ffdf1

File: ./contracts/RoyaltiesReceiver.sol
SHA3: d0424b8f878fcf49b9f570099be7d14ea785d9cd24eaa38c3a8b3f873c084782

File: ./contracts/StorageContract.sol
SHA3: 481cf386c217488088ecbf36108068311db90f4e1186733c6b062995393a0a1a

Fourth review scope

Repository:
https://gitlab.com/nomadhub/smart-contracts
Commit:
45f2029531e2965c648d8b425efd55c8c52b4cd3
Documentation:
https://gitlab.com/nomadhub/smart-contracts/README.md
Integration and Unit Tests: Yes
Contracts:
File: ./contracts/Factory.sol
SHA3: f349766417a36dde47a0e8f7fc8a4dd1bc8db58cebdb36b9e72c7be82f90c080

```
File: ./contracts/interfaces/IFactory.sol
SHA3: 30530929623b49eeaacf25917cba356caff3863bd295dae47d81acb2f8160ca3

File: ./contracts/interfaces/IStorageContract.sol
SHA3: 39634a25012b991dad7b952ad6802d577261a554850feb13bfed9798508783ab

File: ./contracts/NFT.sol
SHA3: df29e49ccd560bef2287ed59823067d178f3bdccbaf69080a65d4790b69ca5c8

File: ./contracts/ReceiverFactory.sol
SHA3: 3f9dd893eb4619910733f4d50c6fafa2666edddb2f6d74b05ea5b90ea25ffdf1

File: ./contracts/RoyaltiesReceiver.sol
SHA3: f8c3f3451e4c2c2b949c89d3bcadd4a6ff6367ab8fb0f99158173fb2817d2f65

File: ./contracts/StorageContract.sol
SHA3: e2b42e5c520b03bd00facf39015143b6b4d2275a63c73edd30730edf2eddc9bc
```

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

Executive Summary

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

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code follows the style guide.

Test coverage

Test coverage of the project is **100.00%**.

- Deployment and basic user interactions are covered with tests.
- All statements and branches are covered.

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: **10.0**.

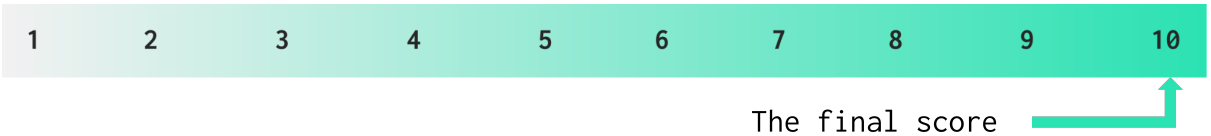

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
17 October 2022	8	5	6	0
08 November 2022	1	0	2	0
22 November 2022	0	0	0	0
08 December 2022	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items 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 required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Passed
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 SWC-126	All external calls should be performed only to trusted addresses.	Passed
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.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
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.	Passed
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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

The protocol allows users to create their own NFT collection, whose tokens represent invitations to the corresponding hub (community). All the collections are deployed via the Factory contract. Users must specify the name, the symbol, contractURI, paying token address, mint price, whitelist mint price, max collection size, and the flag, which shows if NFTs of the collection will be transferable or not. The name, symbol, contractURI, and other parameters (such as royalties size and receiver) need to be moderated on the backend, so BE's signature will be needed for the collection deployment. The factory implementation can be changed, so the information about deployed collections is stored in a separate Storage contract. Factory will be deployed via proxy.
The files in the scope:

- Factory.sol - contract responsible for producing the new instance with defined name and symbol.
- NFT.sol - contract that allows to mint ERC721 tokens with different payment options and security advancements.
- ReceiverFactory.sol - creates an instance of `RoyaltiesReceiver` where each account in `payees` is assigned the number of shares at the matching position in the `shares` array.
- RoyaltiesReceiver.sol - initiates an instance of `RoyaltiesReceiver` where each account in `payees` is assigned the number of shares at the matching position in the `shares` array.
- StorageContract.sol - stores the information about deployed collections.
- IFactory.sol - interface for Factory.sol.
- IStorageContract.sol - interface for StorageContract.sol.

Privileged roles

- Factory.sol roles:
  - Owner - can set platform commission, platform address and signer
  - Signer - address used for validation of signatures
- NFT.sol roles:
  - Creator - can set paying token and mint prices
- RoyaltiesReceiver.sol roles:
  - Payee - address which will receive number of shares at the matching position in the `shares` array
- StorageContract.sol roles:
  - Owner - allowed to set factory address
  - Factory - allowed to add new instance

Risks

- Reviewed contracts are upgradable but are safe to use only if they are used as a first implementation. Otherwise, it will not be verified that the upgrade is done properly due to the lack of information provided by the Customer.
- NFT creators can choose the payment token between ETH or any ERC20. Existence of an ERC20 token at the selected contract address is not verified.
- Project relays on off-chains logic and validations.
- In case payee will be the contract without receive/fallback function or contract with revert on fallback call, all payees would not receive payment, and funds would be locked on the contract.
- The project is enforcing the OpenSea on-chain creator fees system as they plan to be using OpenSea as their only reselling MarketPlace.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. Race Condition

Functions *setMintPrice* and *setWhitelistMintPrice* allow creators to change the price of assets instantly. In case when the user does not check the new minting price, he can spend more funds than expected.

Any type of race conditions should not be possible.

Users can spend more funds than expected.

Path: ./contracts/NFT.sol : function mint()

Recommendation: add an additional argument to function mint with a price of the asset to validate an actual price.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

2. Funds Lock

Contract implements receive function without the possibility to withdraw directly transferred funds.

Native coins and tokens should have mechanisms for their withdrawal from the contract if they are accepted by the contract.

Path: ./contracts/NFT.sol : function receive()

Recommendation: remove *function receive* from NFT.sol contract or add documentation of why it is needed and implement proper functionality to withdraw funds from contract.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

3. Funds Lock

Function mint does not implement checks for the amount of native chain currency. It is possible to pay more funds than regular or whitelisted prices.

Excess funds would be locked in the contract.

Path: ./contracts/NFT.sol : function mint()

Recommendation: check the exact amount of Ether sent with the price of the asset.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

4. **Denial of Service Vulnerability; Funds Lock**

   Sending Ether to another address in Ethereum involves a call to the receiving entity. There are several reasons why this external call could fail.

   In case payee will be the contract without receive/fallback function or contract with revert on fallback call, all payees would not receive payment, and funds would be locked on the contract.

   Additionally, distribution of ERC20 token can reach block Gas limit in case there are too many payees.

   Path: ./contracts/RoyaltiesReceiver.sol : functions releaseAll(), releaseAll(IERC20 token)

   Recommendation: replace method with pull pattern to methods which will allow to withdraw funds with pull pattern to single payee to avoid risks.

   Status: Mitigated (There is no such option that the receiver wouldn't be able to receive ETH, because all wallets are validated on the BE)

5. **Signed Message Replay Attack**

   The project does not use chain identifiers for signatures and validation.

   Chain identifiers should always be used.

   Paths: ./contracts/Factory.sol : function _verifySignature()

   ./contracts/NFT.sol : function mint()

   Recommendation: use EIP712 from OpenZeppelin contracts for signature verification.

   Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

6. **Undocumented Behaviour**

   According to the documentation, *feeReceiver* should be responsible for splitting all payments between creator and platform address. In current implementation, all mint commissions are split directly with the *mint* function.

   Path: ./contracts/NFT.sol : function mint()

   Recommendation: if this is desired logic, then implementation of ERC2981Upgradeable.sol can be removed from the contract as it is not used in the proper way.

   Status: Mitigated (feeReceiver is only responsible for distributing royalties from secondary sales. ERC2981Upgradeable shows marketplaces (which support it), royalties information)

7. Race Condition

If the payment token is changed before the new price of the minting is settled, it is possible to benefit from the Race Condition.

Example:

A user received a signature for minting desired NFT. The paying token is set to ETH and the price is 1e18. The contract creator decided to change the payment token to USDC and the price to 2000 USDC (token decimals 1e18). After the creator updates the payment token to USDC and before he sets a new price, the user can mint desired NFT for only 1 USDC.

Path: ./contracts/NFT.sol : function mint()

Recommendation: add paying token address and minting price from contract storage to signature or merge the functions *setPayingToken* and *setMintPrice,* so they are changed simultaneously.

Status: Fixed (45f2029531e2965c648d8b425efd55c8c52b4cd3)

Medium

1. Unchecked Token Transfer

ERC20 transfer functions return bool after transfers, and it is important to implement a return value check for this return value. This issue leads to unintended behavior of the contract regarding token transfer results.

Path: ./contracts/NFT.sol : function mint()

Recommendation: implement a return value check for token transfers.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

2. Missing Events Emit on Changing Important Values

It is recommended to emit events after changing important values. This will make it easy for everyone to notice such changes.

Paths: ./contracts/StorageContract.sol : function setFactory(), addInstance()

./contracts/Factory.sol : function setPlatformCommission(), setPlatformAddress(), setSigner()

Recommendation: implement event emits after changing contract values.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

3. Transfer Function Fail Risk

If receiver is a contract address, its code (more specifically: its Receive Ether Function, if present, or otherwise its Fallback Function, if present) will be executed together with the transfer call (this is a feature of the EVM and cannot be prevented). If that execution runs out of Gas or fails in any way, the Ether transfer

will be reverted, and the current contract will stop with an exception. Future changes in the Gas costs for some opcodes can break this design.

Path: ./contracts/NFT.sol : function mint()

Recommendation: implement sending payments with the function call() and check return value.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

4. **Uninitialized Implementation**

Contract ERC2981Upgradeable.sol inherited in the NFT.sol, but is not initialized.

Path: ./contracts/NFT.sol

Recommendation: ERC2981Upgradeable should be initialized.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

Low

1. **Assert Violation**

Properly functioning code should never reach a failing assert statement.

Paths: ./contracts/StorageContract.sol : function getInstanceInfo()

./contracts/RoyaltiesReceiver.sol : function payee()

Recommendation: add a check for correct instance Id and payee index.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

2. **Missing Zero Address Validation**

Address parameters are being used without checking against the possibility of 0x0.

Paths: ./contracts/StorageContract.sol : function setFactory()

./contracts/Factory.sol : function setPlatformAddress(), setSigner(), initialize(), produce()

./contracts/NFT.sol : setPayingToken(), initialize()

Recommendation: implement zero address checks.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

3. **Functions that Can Be Declared External**

“public” functions that are never called by the contract should be declared “external” to save Gas.

Paths: ./contracts/Factory.sol : function produce()

./contracts/NFT.sol : function initialize()

./contracts/ReceiverFactory.sol : deployReceiver()

./contracts/RoyaltiesReceiver.sol : totalShares(), shares(), payee()

./contracts/StorageContract.sol : getInstanceInfo()

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

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

4. **Style Guide Violation**

Functions should be grouped according to their visibility and ordered:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

Within a grouping, place the view and pure functions last.

Inside each contract use the following order:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

The modifier order for a function should be:

1. Visibility
2. Mutability
3. Virtual
4. Override
5. Custom modifiers

Remove additional white lines.

Do not put underscore for public function.

The provided projects should follow the official guidelines.

Paths: ./contracts/Factory.sol, ./contracts/NFT.sol, ./contracts/RoyaltiesReceiver.sol, ./contracts/StorageContract.sol, **./**contracts/interfaces/IStorageContract.sol

Remediation: for IStorageContract.sol the correct order of function mutability was not provided.

Recommendation: follow the official Solidity guidelines.

Status: Fixed (45f2029531e2965c648d8b425efd55c8c52b4cd3)

5. **Reading State Variables in a Loop**

   Reading a state variable or an attribute of it may be costly, in terms of Gas fees. The variable _payees is being read in loops.

   Path: ./contracts/RoyaltiesReceiver.sol : functions releaseAll()

   Remediation: the maximum limit for royalties receivers was limited to 2.

   Recommendation: save the state variable or its attribute into a local variable.

   Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

6. **Duplicate Code**

   Duplication of code may lead to unnecessary Gas consumption. There is a duplication in *setPayingToken*, *setMintPrice* and *setWhitelistMintPrice*. The three functions use the same *require*.

   Path: ./contracts/NFT.sol : function setPayingToken(), setMintPrice (), setWhitelistMintPrice()

   Recommendation: replace duplication with a modifier.

   Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

7. **Typos in Documentation**

   There is a typo in “1.1 Roles” of the documentation. It says *approvement*, however, it should be *approval*.

   There is a typo in “1.3 Use Cases” of the documentation. It says *instanses*, however, it should be *instances*.

   There is a typo in “1.3 Mint token from the collection” of the documentation. It says *immediatelly*, however, it should be *immediately*.

   There is a typo in “2.2.2 Factory.sol” of the documentation. It says *comission*, however, it should be *commission*.

   There is a typo in “2.2.4 RoyaltiesReceiver.sol” of the documentation. It says *changes*, however, it should be *change*.

   There is a wrong numeration from 2.2.3 until 2.2.5.

   Recommendation: fix typos.

   Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

8. **Unindexed Events**

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Paths: ./contracts/NFT.sol : event EthReceived

./contracts/ReceiverFactory.sol : event ReceiverCreated

./contracts/RoyaltiesReceiver.sol : events PayeeAdded, PaymentReleased, ERC20PaymentReleased, PaymentReceived

Recommendation: use the “indexed” keyword for the event parameters.

Status: Fixed (b8946732a842f85c71476242ff39a850c5615664)

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.