HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Formless
Date: April 04th, 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 Formless.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC721 token; ERC1155 token; Factory
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.formless.me
Timeline	10.03.2022 - 04.04.2022
Changelog	18.03.2022 - Initial Review 04.04.2022 - Revise

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 Formless (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/FormLess-Games/Formless-Official-Market
Commit:
036a2f77ab961ecbd319a75370e9fa419ece9bf9
Technical Documentation: Yes (https://formless.gitbook.io/white-paper-eng/)
JS tests: No
Contracts:
NFTFactory.sol
Template/InitializableERC1155.sol
Template/InitializableERC721.sol
Template/BlindBoxV2.sol
lib/CloneFactory.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 superficial functional requirements and no technical requirements. The total Documentation Quality score is **6** out of **10**.

Code quality

The total CodeQuality score is **3** out of **10**. Long lines. Missed NatSpec. No Unit Tests. The code is messy. Too many excess state variable access. Boolean equality.

Architecture quality

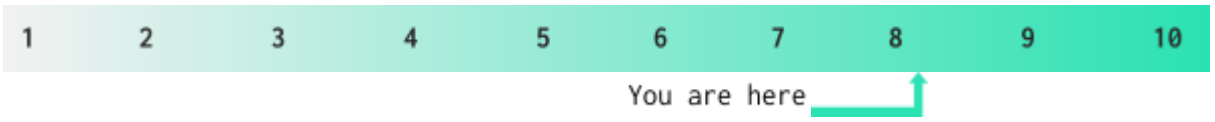
The architecture quality score is **3** out of **10**. All the logic is implemented in template files. Some logic commented-out. Hard to find and understand a straightforward logic.

Security score

As a result of the audit, security engineers found **3** low severity issues. The security score is **10** out of **10**. All found issues are displayed in the “Issues overview” section.

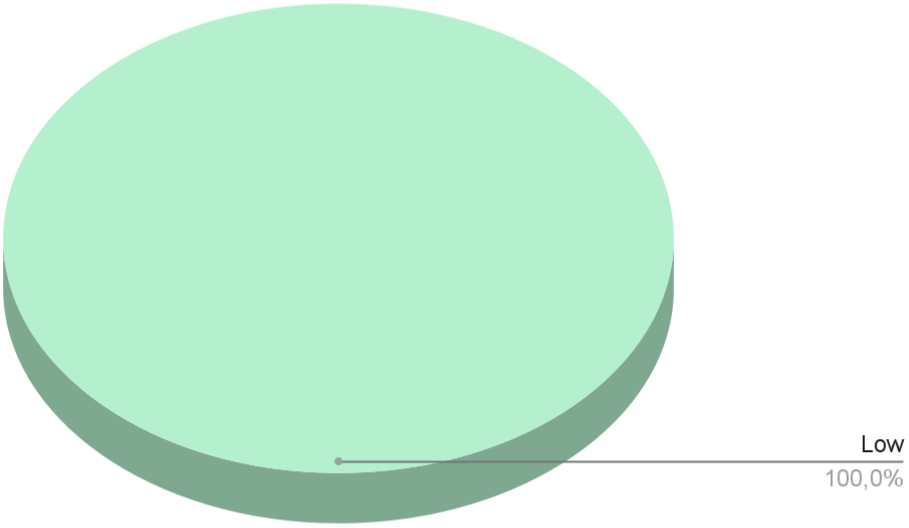
Summary

According to the assessment, the Customer's smart contract has the following score: **8.2**



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

No critical severity issues were found.

High

No high severity issues were found.

Medium

Using the `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: BlindBoxV2.sol

Functions: mint

Recommendation: Use the following construction instead:

```
payable(_saleConfig.treasury).call(msg.value)("")
```

Status: Fixed (Revised Commit: 036a2f7)

Low

1. **Unused variable.**

   The state variable `nextTokenId` is not used in the contract. It is neither initialized nor accessed anywhere. It is always ZERO.

   Contract: InitializableERC721.sol

   Variable: nextTokenId

   Recommendation: remove unused variable.

   Status: Fixed (Revised Commit: 036a2f7)

2. **State variables that could be declared constant.**

   Constant state variables should be declared constant to save gas.

   Contract: InitializableERC1155.sol

   Variable: _baseUri

   Recommendation: add the **constant** attribute to state variables that never change.

   Status: Fixed (Revised Commit: 036a2f7)

3. **Reading state variable in the loop.**

   It could be costly to read the state in the loop in a gas matter.

   Contract: BlindBoxV2.sol

Function: initialize

Recommendation: consider not reading the `erc20Tokens[i]` after storing the value. Read the local `_erc20Tokens[i]` instead.

Status: Not fully fixed. Still reading the state on line#107 (Revised Commit: 036a2f7)

4. **Reading state variable in the loop.**

   Using the state variable as the condition statement in the for-loop causes accessing it every loop cycle.

   Contract: BlindBoxV2.sol

   Function: open

   Recommendation: read the `perSupplys.length` into the local memory variable before line#203 and use it afterward. Also, read the `perSupplys[i]` right before the second loop and use the local variable in the conditions statement.

   Use the same recommendations for the `erc20Tokens` variable in the same function.

   Status: Not fully fixed. Still reading the length in the loop on line#213 (Revised Commit: 036a2f7)

5. **Boolean equality.**

   Boolean constants can be used directly and do not need to be compared to **true** or **false**.

   Contract: BlindBoxV2.sol

   Function: open

   Recommendation: remove the equality to the boolean constant.

   Status: Not fixed (Revised Commit: 036a2f7)

6. **Unused variable.**

   `baseURI` and `boxURI` state variables are assigned by the `setBaseURI` and `setBoxURI` functions, respectively but are never used in the contract.

   Contract: BlindBoxV2.sol

   Variable: baseURI, boxURI

   Recommendation: remove the variable or update usages of those.

   Status: Fixed (Revised Commit: 036a2f7)

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

Recommendations

1. Clean up the code. Follow the Solidity Style Guide for naming, lines length, code layout.
2. Check the code for a gas-efficiency. There are a lot of excess state read and writes.

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 it 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.