HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: With You Lab
Date: Dec 08th, 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 With You Lab
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Non-fungible token
Platform	EVM
Language	Solidity
Methodology	Link

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 With You Lab (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:

Review scope

Repository	https://github.com/withyou-labs/ERC721BO
Commit	6e7f16c
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Assets.sol SHA3: e3d8e4b4cd732f10c2aa36223d7f504a1f7eceaf5c645feb84d47e871b283a4b File: ./contracts/Bits.sol SHA3: fc4860c823cc8f407b7fa778bffba39545d0cfe1a9aa964c29d573598c2a2cb4 File: ./contracts/Counter.sol SHA3: d501836c1535585646a78ed1df1af45812bf5d17876eb1294267e8b2227cc2b5 File: ./contracts/ERC721BO.sol SHA3: ef8475158302558ad48ff779ebd07ae0eeefdf082fbea3258d575fcbb2a1b876 File: ./contracts/extensions/ERC721BONonburnable.sol SHA3: 2727a7441242626c327334962c672b4c014e30efa63332ed10d5dcfd9dad8922 File: ./contracts/IERC721BO.sol SHA3: 88231de0f770b7e7f152b3a3dfe2fc83cef7bbe49af9920732825013df703177 File: ./contracts/Owners.sol SHA3: 60521320e6b49a7832bb0dccf5e241efecc611c18f3b4c20822c5b95a8624460

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 the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations, but cannot lead to assets loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization. These issues won't have a significant impact on code execution, but affect the code quality

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 official Solidity style guides.

Test coverage

Test coverage of the project is **99.04%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Positive and negative cases are covered.
- Interactions by several users are tested.

Security score

As a result of the audit, the code does not contain issues. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**.

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
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.	Passed
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.	Passed
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	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 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.	Passed

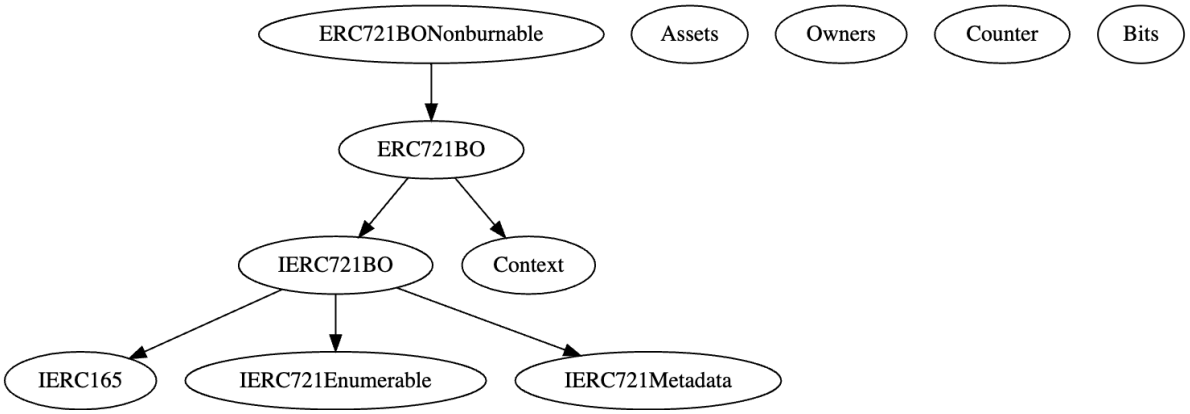
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	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 sufficient, 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, which may be changed in the future.	Passed

System Overview

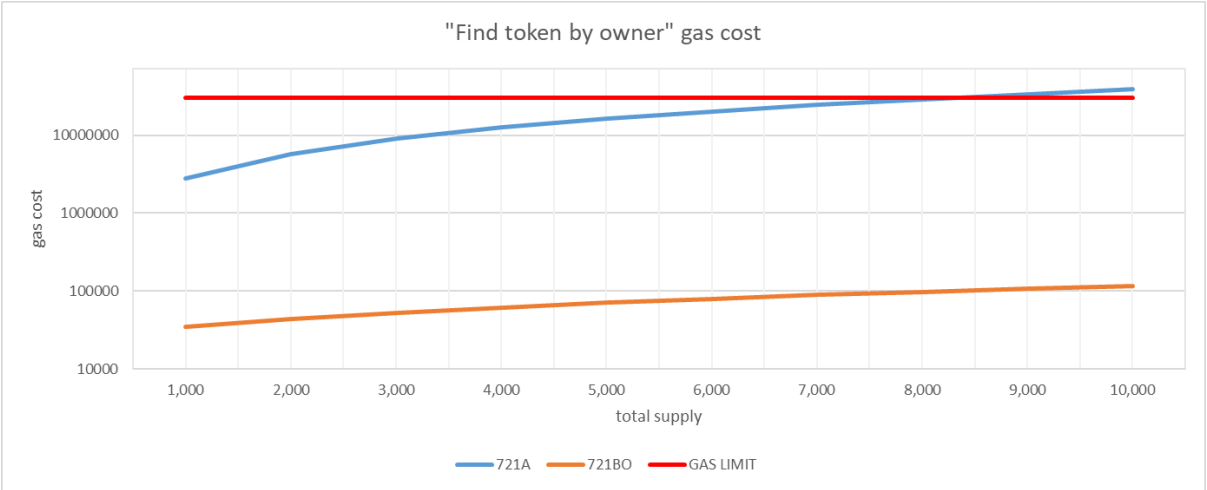
ERC721BO is a smart contract development library for creating and managing multiple NFTs in a single transaction while maintaining compatibility with ERC721Enumerable. This library implements the IERC721Enumerable interface. However, unlike typical ERC721 and ERC721Enumerable implementations, it employs an ERC721A-inspired lazy initialization mechanism and flag ownership management mechanism.
Following contracts are:

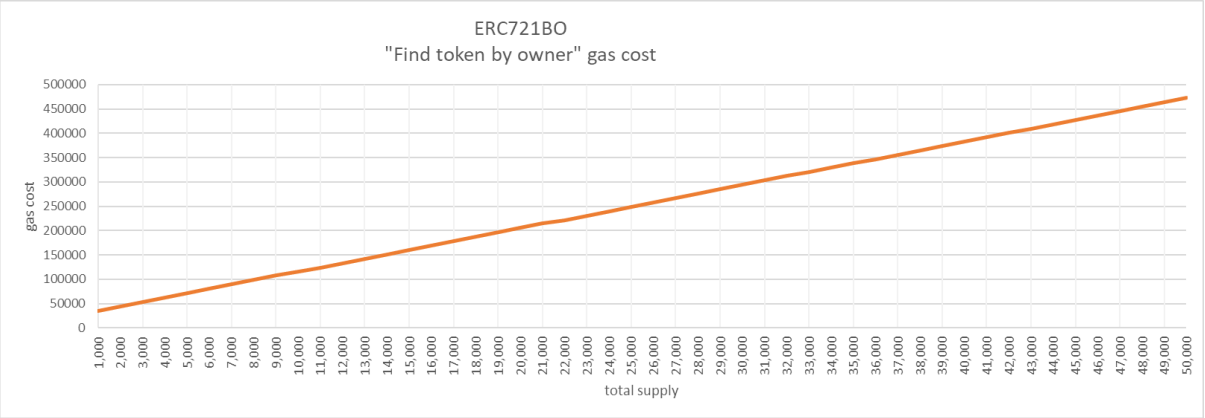
- *ERC721BO.sol* - library for creating and managing multiple NFTs.
- *ERC721BONonburnable.sol* - abstract contract for non-burnable ERC-721.
- *IERC721BO.sol* - interface for ERC721BO.
- *Assets.sol* - library for ownership flag table.
- *Bits.sol* - library for work with bits.
- *Counter.sol* - library for counter with initial value of non-zero.
- *Owners.sol* - library for owner list using delayed initialization mechanism.

Inheritance graph of ERC721BO contract:



Libraries Assets, Bits, Counter, Owners are used on ERC721BO smart contract.
Gas consumption benchmarks (provided by the Customer):





Risks

- Gas consumption of the contract grows with the number of minted tokens. Though the maximum supply is limited to 1 << 16 (65536) NFTs, and the system can operate on those numbers, the overall growth of gas consumption should be considered.

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

No medium severity issues were found.

Low

No low severity issues were found.

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.