SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Museum of War NFT

Date: May 5th, 2022

This document 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 containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Museum of War NFT.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC721 token; ERC1155 token; Airdrop; Tokensale; Merger;
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://metahistory.gallery
Timeline	30.04.2022 - 06.05.2022
Changelog	03.05.2022 - Initial Review 06.05.2022 - Second Review

Table of contents

Introduction 4

Scope 4

Severity Definitions 5

Executive Summary 6

Checked Items 7

System Overview 10

Findings 11

Disclaimers 13

Introduction

Hacken OÜ (Consultant) was contracted by Museum of War NFT (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/museum-of-war/nft/
Commit:
bd9d485bc7040b988404d94a575a058bccc7cf0d
Technical Documentation:
Type: Superficial functional and technical description
Link: https://github.com/museum-of-war/nft/blob/master/docs/

JS tests: Yes
Contracts:
File: ./contracts/CollectionMH.sol
SHA3: 2508fbe83e5a44094f2f7a256a86572dbdd3ae2728242006bad2fa10f649e0d0

File: ./contracts/DropMH.sol
SHA3: 6e083d89df95238769d5dae238d65e845db080f41022366f39303f39b473758a

File: ./contracts/MergerMH.sol
SHA3: fe3d12bdac373b478cd0eb176347eddac87ae4bf9d91268e12d5ea3e884fecd7

Second review scope

Repository:
https://github.com/museum-of-war/nft/
Commit:
17d73efc00269afebaf3031f1671a1e3db3df49f
Technical Documentation:
Type: Superficial functional and technical description
Link: https://github.com/museum-of-war/nft/blob/master/docs/

JS tests: Yes
Contracts:
File: ./contracts/CollectionMH.sol
SHA3: 2508fbe83e5a44094f2f7a256a86572dbdd3ae2728242006bad2fa10f649e0d0

File: ./contracts/DropMH.sol
SHA3: aefaff57a8d541634e40e65b8ede372691fbf6f0f8b24875530055e73fa8b43a

File: ./contracts/MergerMH.sol
SHA3: 0058a76a5dfdd1b8d2ca58ed76257e70fbf57ff9473cc1a9a3a6ce730e585d26

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 measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided superficial functional and technical requirements. The total Documentation Quality score is **7** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. The code follows best practices. Unit tests were provided.

Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clean and clear.

Security score

As a result of the audit, security engineers found **2** high and **3** low severity issues. The security score is **0** out of **10**.

As a result of the second review, security engineers found **no** new 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: **9.7**.

1
2
3
4
5
6
7
8
9
10
You are here

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.	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.	Not Relevant
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 destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Not Relevant
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
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 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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	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
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
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

System Overview

- *CollectionMH* – ERC721 token contract with batch minting to the receiver in the constructor.
- *DropMH* – ERC1155 token contract with sale and airdrop functionality.
- *MergerMH* – ERC721 token contract used to merge 2 tokens of the same event into one token of the next level. This contract gives rewards for levels, starting from the 2nd.

Privileged roles

- The owner of *CollectionMH* and *MergerMH* can modify the base URI and can lock metadata forever.
- The owner of *DropMH* can modify the base URI, lock metadata forever, pause/unpause minting and transferring, change burner address, and change the maximum number of mints per wallet.
- The burner address of *DropMH* can burn tokens from any account.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

1. Highly permissive owner access to burn tokens.

The owner of the contract can set an address that will have the ability to burn tokens of any user.

This can lead to the loss of user tokens if the owner's account falls into the hands of an attacker.

Contract: DropMH.sol

Functions: setBurner, burn

Recommendation: remove the ability to burn user tokens or restrict the ability to set any address as a burner. In the latter case, the burner must be the contract with clear burning rules.

Status: Fixed (second review)

2. Highly permissive owner access to stop token transfers.

The contract owner may pause the contract, which will stop the transfer of all tokens.

This can lead to the blocking of user tokens if the owner's account falls into the hands of an attacker.

Contract: DropMH.sol, ERC1155Pausable.sol

Functions: pause, _beforeTokenTransfer

Recommendation: remove the ability to block token transfers.

Status: Fixed (second review)

🟩🟩 Medium

No medium severity issues were found.

🟦 Low

1. Unused import.

The contract imports Pausable.sol but never uses it.

This is against best practices.

Contract: DropMH.sol

Recommendation: remove unused import.

Status: Fixed (second review)

2. Commented code.

The contract contains a lot of commented code.

This impairs the readability of the code.

Contract: MergerMH.sol

Recommendation: remove commented code.

Status: Fixed (second review)

3. **Irrelevant code.**

The rewardsCount variable and its calculations have no impact on the contract execution.

This impairs the readability of the code and causes an increase in computations (and unnecessary Gas consumption).

Contract: MergerMH.sol

Function: mergeBaseBatch

Recommendation: remove unneeded code.

Status: Fixed (second review)

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 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.

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.