Blaize.Security

March 14th, 2022 / V. 2.0

NEMUS NFT
SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Nemus contract's source code was taken from the repository provided by the Nemus team.

SCORE **9.8**/10

The scope of the project is **Nemus** set of contracts**:**

1/ contracts/nft/NeaNFT.sol

2/ contracts/nft/ERC721A.sol

Contracts repository:
https://github.com/Nemus-Team/nemus-contracts

Initial commit:

- 0a160065f10347e3f263c901fe49bcfbf1f89daa

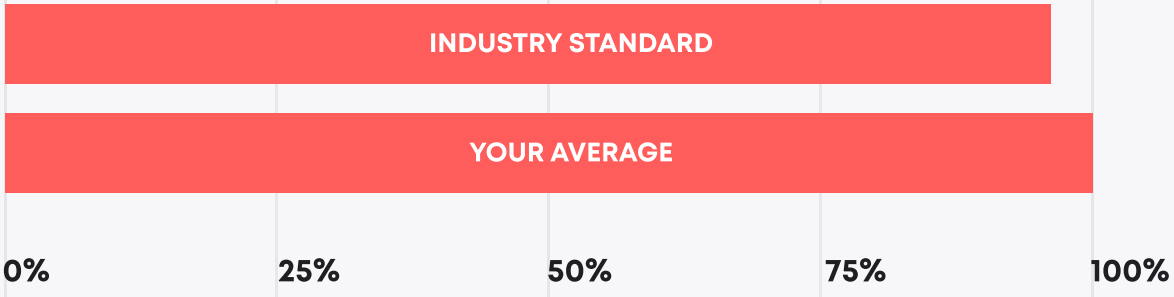
Final commit:

- 9b8a4c325a86c696f0498e07db12c1e2ae55e29c

TECHNICAL SUMMARY

In this report, we consider the security of the contracts for Nemus protocol. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of **Nemus** smart contracts conducted between **February 7th, 2022 - March 14th, 2022.**

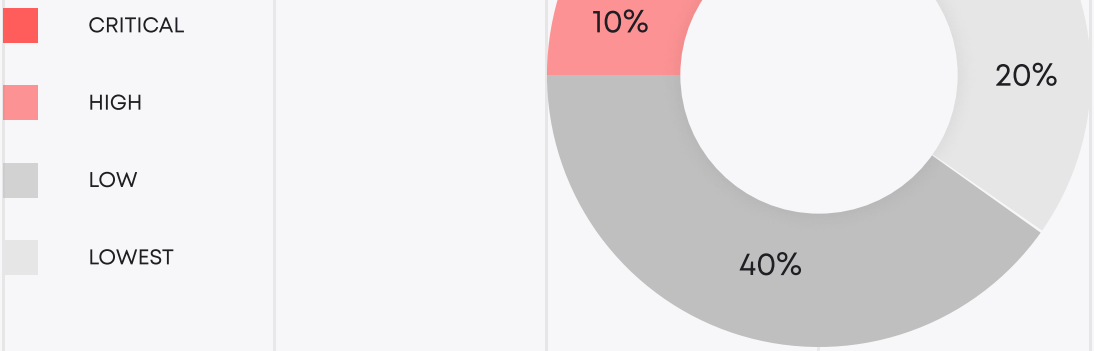
Testable code



The testable code is 95.2%, which is above the industry standard of 95%.

The scope of the audit includes the unit test coverage, that bases on the smart contracts code, documentation and requirements presented by the Nemus team. Coverage is calculated based on the set of Truffle framework tests and scripts from additional testing strategies. Though, in order to ensure a security of the contract Blaize.Security team recommends the Nemus team put in place a bug bounty program to encourage further and active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of found issues and their severity. A total of 10 problems were found. 10 issues were fixed or verified by the Nemus team.

	FOUND	FIXED/VERIFIED
Critical	3	3
High	1	1
Medium	0	0
Low	5	4
Lowest	2	2

SEVERITY DEFINITION

Critical

A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.

High

A system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge information or financial leak. Needs immediate improvements and further checking.

Medium

A system contains issues which may lead to mediumfinancial loss or users' private information leak. Needs immediate improvements and further checking.

Low

A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.

Lowest

A system does not contain any issue critical to the secure work of the system, yet is relevant for best

AUDITING STRATEGYAND TECHNIQUES APPLIED \ PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

Automated analysis:

Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. Checking smart contract logic and comparing it with the one described in the documentation.

EXECUTIVE SUMMARY

The contract contained critical issue from the standard auditors checklist together with several issues with NFT minting. Though, the team has fixed the issue.

All other issues were connected to the code quality and gas optimizations. The contract was represented as the custom implementation of the ERC721 contract with unoptimal code with quite low code quality. Nevetheless, during the audit, Nemus team significantly increased the quality of the codebase, restored the contract functionality and provided appropriate comments to the functionality.

	RATING
Security	9.8
Gas usage and logic optimization	9.6
Code quality	10
Test coverage**	10
Total	9.8

** Contracts have good native coverage which was checked within the scope of the audit. Nevertheless - security team has prepared own set of tests.

COMPLETE ANALYSIS

CRITICAL	✔ Resolved

tx.origin usage.

NeaNFT.sol, redeem()
The function utilizes the comparison against tx.origin, which is first of all forbidden within the standard auditors list, and actually does not give the protection against call from the contract. So it is recommended to use the isContract() check from the standard Address.sol contract in order to prevent call from the contract.

Recommendation:

Remove tx.origin usage.

Post-audit.

Usage of tx.origin was removed from the contract.

CRITICAL	✔ Resolved

Unverified override.

NeaNFT.sol, _mint()
The function overrides standard mint functionality from ERC721 and omits all security checks with no reason.

Recommendation:

Remove unnecessary override.

Post-audit.

Function was replaced with ERC721A._safeMint()

CRITICAL ✔ Resolved

Incorrect minting by ID.

NeaNFT.sol, _mint()
ERC721.sol, _mint()
Functionality is aimed to mint a certain NFT id to the user, though the ID actually minted will not be the same as added for the user. The array of owners of ids is not synchronized with the actually minted ids.

Recommendation:

Restore standard ERC721 contract, or synchronize minting with the owners array, where ids are stored. For now neither generated event, no checks against the ids to be inline, nor minting of the next if are not synchronized with what the user will receive. In general, minting functionality should be re-verified.

Post-audit.

Function was replaced with ERC721A._safeMint().

HIGH ✔ Resolved

Incorrect burn functionality.

ERC721.sol: _burn()
Burn function works incorrectly, because of several reasons:
it actually does not change the supply of the tokens, and this pitfall is achieved in NeaNFT.sol redeem() function, where token supply is checked;
In combination with _mint() function it allows to mint token with place on the wrong id in case of burnt token re-mint;
There is no any check against burnt token in the contracts set;
And actually burn functionality is not used throughout the contracts set.
So, since this functionality is not needed it is recommended to remove it, as it may influence further development, or to use the standard implementation of ERC721, since the problem is in the changed storage access.

Recommendation:

Remove burn functionality or restore the standard contract for ERC721.

Post-audit.

Function was removed.

LOW ✔ Resolved

Unlimited cycle.

ERC721.sol: Function balanceOf() has unlimited cycles - since there are no restrictions on the amount of tokens minted, view calls to this function may fail.

Recommendation:

Add another view function to check the balance of the owner within the range of ids.

Post-audit.

Function was replaced with ERC721A.balanceOf() which has no unlimited cycle.

LOW ✔ Resolved

Unnecessary loop for searching owner address.

ERC721A.sol: function ownershipOf().
The purpose of the function is to return the TokenOwnership struct of ‘tokenId”.
It is enough to return _ownerships[tokenId] instead of iterating through mapping.

Recommendation:

Remove loop and return _ownerships[tokenId].

LOW	✔ Resolved

Validate function parameters.

NeaNFT.sol: function setSaleData().
Parameters should be validated not to be zero, '_presaleStart' should be greater than block.timestamp and '_publicStart' should be greater than '_presaleStart'.

Recommendation:

Validate parameters.

LOW	✔ Resolved

Missing default visibility.

NeaNFT.sol: explorationAddress and conservationAddress have visibility defined.

Recommendation:

Add public/private qualificator for the variables .

LOW	Unresolved

Unused internal function.

There is the _numberMinted() function (line 148) that is used nowhere in ERC721A.sol.

Recommendation:

Remove the unused function.

LOWEST	✔ Resolved

Call to totalSupply.

NeaMint.sol, redeem(). The function calls directly to the storage, though the totalSupply() method from ERC721ENumerable can be used for the encapsulation.

Recommendation:

Use existing function.

LOWEST	✔ Resolved

Modified ERC721.

ERC721.sol
This contract is the modified OpenZeppelin version, with deleted storage for balances. Actually this modification has no pros against the standard implementation, because it does not simplify or optimize the solution. In general it is recommended to use the OpenZeppelin version of the functionality unless there are breaking changes to the core of the NFT contract.

Recommendation:

Use the standard version of the contract.

Post-audit.

Contract was replaced with ERC721A.sol which has significant changes compared to OpenZeppelin version.

	ERC721A.sol	NeaNFT.sol
✔ Re-entrancy	Pass	Pass
✔ Access Management Hierarchy	Pass	Pass
✔ Arithmetic Over/Under Flows	Pass	Pass
✔ Delegatecall Unexpected Ether	Pass	Pass
✔ Default Public Visibility	Pass	Pass
✔ Hidden Malicious Code	Pass	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass	Pass
✔ External Contract Referencing	Pass	Pass
✔ Short Address/ Parameter Attack	Pass	Pass
✔ Unchecked CALL Return Values	Pass	Pass
✔ Race Conditions / Front Running	Pass	Pass
✔ General Denial Of Service (DOS)	Pass	Pass
✔ Uninitialized Storage Pointers	Pass	Pass
✔ Floating Points and Precision	Pass	Pass
✔ Tx.Origin Authentication	Pass	Pass
✔ Signatures Replay	Pass	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Contract: NeaNFT

- ✓ Token owner should not transfer a token if a set token mode (789ms)
- ✓ Owner should add addresses to the allow list (130ms)

Redemption

- ✓ User should redeem (301ms)
- ✓ User should not redeem during early access if he is not on the allowed list (79ms)
- ✓ User should redeem during early access if he is on the allowed list (335ms)
- ✓ User should not redeem if zero amount (45ms)
- ✓ User should not redeem if not enough amount

Setters

- ✓ Owner should set a base URI
- ✓ Owner should set an exploration contract address
- ✓ Owner should not set an exploration contract address if zero address
- ✓ Owner should set an conservation contract address
- ✓ Owner should not set an conservation contract address if zero address
- ✓ Owner should set an allowance of token mode setting (41ms)
- ✓ Owner should set an address of a Nea mint ticket factory (100ms)
- ✓ Owner should not set an address of a Nea mint ticket factory if zero address

- ✓ Owner should set an early access end time
- ✓ Owner should not set an early access end time if past time
- ✓ Owner should batch set tiers of tokens
- ✓ Owner should not batch set tiers of tokens if array length mismatch

Setting of a token mode

- ✓ Owner should set the token exploration mode (38ms)
- ✓ Token owner should set the exploration mode for his token (138ms)
- ✓ Owner should set the token conservation mode
- ✓ Owner should set the token combo mode (45ms)
- ✓ Owner should return to the mode-free state of a token (59ms)
- ✓ Contract, token non-owner can not set a token mode (123ms)
- ✓ Owner should not set if the same token mode
- ✓ Owner should not set if no allowance of token mode setting

Getters

- ✓ Should get token's owner data
- ✓ Should get token's ticket size ID
- ✓ Should get token's mode
- ✓ Should get token's tier
- ✓ Should get owner's token IDs
- ✓ Should get an empty array if empty wallet

Non-owner can not

- ✓ add addresses to the allow list
- ✓ set a base URI
- ✓ set an address of a Nea mint ticket factory
- ✓ set an exploration contract address
- ✓ set an conservation contract address
- ✓ set an early access end time
- ✓ set an allowance of token mode setting
- ✓ batch set tiers of tokens

- ✓ Should get a balance of an owner (353ms)
- ✓ Should not get a balance if zero address
- ✓ Should get an owner of a token (296ms)
- ✓ Should not get an owner of a nonexistent token

Enumeration extension

- ✓ Should get the total supply (56ms)
- ✓ Should get a token index (42ms)
- ✓ Should not get a token index if global index out of bounds
- ✓ Should get a token ID of an owner at a given index (41ms)
- ✓ Should get a token ID of an owner at a given index when some tokens (52ms)
- ✓ Should get a token ID of an owner at a given index when there are some NFT owners (79ms)
- ✓ Should not get a token ID of an owner by an index if owner's balance is more than the index

Transfer

- ✓ Token owner should transfer a token (63ms)
- ✓ Token owner should transfer a token when some tokens (61ms)
- ✓ Token owner should transfer a token when some tokens with different owners (94ms)
- ✓ User should not transfer if he does not own the token and does not have approval (43ms)
- ✓ Token owner should not transfer if incorrect owner (51ms)
- ✓ Token owner should not transfer if transfer to zero address (45ms)
- ✓ Should get a token URI (351ms)

59 passing (8s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
ERC721A.sol	90.35	76	78.57
NeaNFT.sol	100	100	100
All files	**95.2**	**80.23**	**86.96**

Also it needs to be mentioned, that Nemus has own set of unit tests with quite good quality.
Also, ERC721A contract mostly contains standard ERC721 functionality which was carefully checked against the standard OpeZeppelin implementation.

DISCLAIMER

The information presented in this report is an intellectual property of the customer including all presented documentation, code databases, labels, titles, ways of usage as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else requirements and be fully secure, complete, accurate and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.