QuillAudits

Audit Report
March, 2024

For

NFTSPACE

Table of Content

Executive Summary

Project Name	NFTSpace
Overview	NFTSpace token is ERC20 upgradeable token which mints hardcoded totalsupply to the msg.sender while initializing.
Timeline	22nd February 2024 - 28th February 2024
Updated Code Received	6th March 2024
Second Review	6th March 2024 - 8th March 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the NFTSpace token contract for quality, security, and correctness.
Source Code	*https://github.com/theyashmathur/nftspace*
Branch	Main
Fixed In	9877fc684697443e7f809ec594cf811fa3387e07

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	2	1

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

No issues were found.

Medium Severity Issues

1. Minting totalsupply without decimals

Path

token.sol#L14

Function

initialize()

Description

intialize() mints 1_000_000_000 (totalsupply) to msg.sender. But here 1_000_000_000 is getting specified without decimals.

This can create problem when user/owner will assume they got 1_000_000_000**e18** and they will try to transfer amount with decimals e.g 100e18 (100000000000000000000) but because only 1_000_000_000 got minted to owner's account, it will fail.

It should be changed to 1_000_000_000**e18** if the value without decimals is not the intended one.

Recommendation

Consider changing 1_000_000_000 to 1_000_000_000**e18** while minting.

Status

Resolved

Low Severity Issues

2. Using floating pragma

Path

token.sol

Description

token.sol is not using a fixed solidity version. It's using floating pragma (pragma solidity ^0.8.21), Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version gets selected while deploying a contract which has a higher chance of having bugs in it.

Recommendation

Remove floating pragma (^) and use a specific compiler version with which contracts have been tested.

Status

Resolved

3. _disableInitializers() should be called in the contstructor

Path

token.sol

Description

In the proxy implementations **_disableInitializers()** should be called in the constructors. If not added, malicious users/address can call initialize() on the implementation smart contract where they would be able to set certain things that initialize() allows to set.

The severity of impact depends on the contract logic, but it's a good practice to add **_disableInitializers()** call in the implementation logic.

Recommendation

Consider adding **_disableInitializers()** in constructor as suggested here: **https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers–**

Status

Resolved

Informational Issues

4. Comment can be removed

Path

token.sol

Description

Contract contains "**// Additional functions and logic specific to your token can be added here**" comment. This comment can be removed before deploying as a good practice.

Recommendation

Consider removing the highlighted comment.

Status

Resolved

Functional Tests Cases

Some of the tests performed are mentioned below:

NFTSPACE:

- ✓ Should mint the hardcoded initial supply to the mgs.sender
- ✓ Should be able to transfer the tokens
- ✓ Should be able to give approval to spender
- ✓ Spender should be able to spend the approved token amount
- ✓ Reverts if sender has less balance while transferring
- ✓ Reverts if spender has less allowance while transferring from other address
- ✓ Owner should be able to transfer the ownership

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the NFTSpace codebase. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.In the end NFTSpace Team resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in NFTSpace smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of NFTSpace smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the NFTSpace to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
March, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com