QuillAudits

AUDIT REPORT

December, 2024

For

NodeOps

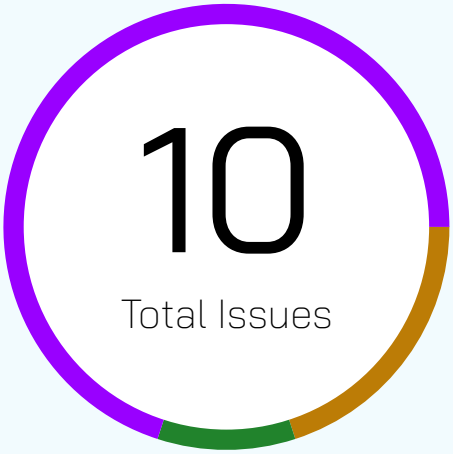
Table of Content

Executive Summary

Project name	NodeOPS
Overview	NodeOPS smart contracts consist of a standard ERC20 token and an airdrop pool contract. The token contract is designed with the additional features that allows whitelisted addresses to freely send and receive these tokens. Non-whitelist token holders can transfer tokens when the transfer feature is enabled. The Airdrop contract is characterized as a pool where the NODE tokens will be deposited by the admin and users can request once to claim their tokens with their derived signatures. The admin can as well send these tokens to users by invoking the privileged transferTo function.
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 NodeOPS Smart Contracts for quality, security, and correctness.
Contracts In Scope	contracts/ERC20/NODE.sol contracts/pools/base/Airdrop.sol contracts/interfaces/IPool.sol contracts/libraries/Roles.sol
Project URL	https://nodeops.xyz/
Commit Hash	258ba8c3d399a9ca673ee70f054f73f28545f3ef
Language	Solidity

Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	11th December 2024 - 13th December 2024
Updated Code Received	13th December 2024
Review 2	16th December 2024
Fixed In	ae0199ed0848d4ead9f05447b4adf346dfb44631

Number of Issues per Severity



High	0 (0.00%)
Medium	2 (20.00%)
Low	1 (10.00%)
Informational	7 (70.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**2**	**1**	**5**
Acknowledged	0	0	0	**2**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ☑ Style guide violation
- ☑ Unsafe type inference
- ☑ Implicit visibility level
- ☑ Gasless send
- ☑ Exception disorder
- ☑ Gas Limit and Loops
- ☑ Timestamp Dependence
- ☑ Re-entrancy
- ☑ Access Management
- ☑ Arbitrary write to storage
- ☑ Centralization of control
- ☑ Ether theft
- ☑ Improper or missing events
- ☑ Logical issues and flaws
- ☑ Using inline assembly
- ☑ Using inline assembly
- ☑ Using throw
- ☑ Upgradeable safety
- ☑ Using delegatecall
- ☑ Using suicide
- ☑ Multiple Sends
- ☑ Revert/require functions
- ☑ Private modifier
- ☑ Missing Zero Address Validation
- ☑ Return values of low-level calls
- ☑ Tautology or contradiction
- ☑ Dangerous strict equalities
- ☑ ERC's conformance
- ☑ Integer overflow/underflow
- ☑ Divide before multiply
- ☑ Address hardcoded
- ☑ Compiler version not fixed

- ☑ Malicious libraries
- ☑ Use of tx.origin
- ☑ SWC Registry
- ☑ Race conditions/front running
- ☑ Arithmetic Computations Correctness

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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Issues

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.

Issue Status

Open

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

Resolved

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

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.

Medium Severity Issues

Add _disableInitializer() in implementation contract's constructor

Resolved

Path

Node.sol

Description

In proxy contracts where there is an initialize() function without access control, it can be front-run by attackers. From the OZ blog:

At the time of deploying the contract, the protocol owner initializes the proxy contract, and the protocol owner becomes the owner of the contract. However, an attacker can call the initialize function on the implementation contract and become the owner of the implementation contract. In this case, the attacker becomes the owner of the implementation contract. If the implementation contract's delegatecall is executed, it could introduce a critical vulnerability. Specifically, the attacker could use delegatecall to execute code from the implementation contract that triggers attack.sol to self-destruct. As a result, all calls made by the proxy contract would fail.

Recommendation

```
constructor() {
_disableInitializers();
}
```

Reverts when DEFAULT_ADMIN_ROLE calls the transfer function to a non-whitelisted address

Resolved

Path

Node.sol

Function

_update()

Description

This implementation would revert when the DEFAULT_ADMIN calls the transfer function with a non-whitelisted recipient. The control flow is:

- allowed will be true.
- enters the next if condition because the recipient is false.
- if the amount to be sent is greater than the transferLimits of from which is 0.
- This reverts with this error message - TransferLimitExceeded

Low Severity Issues

Oversight on the claim function missing the whenNot-Paused modifier

Resolved

Path

Airdrop.sol

Function

claim()

Description

the pausable feature integrated into the contract is to disable and enable all withdrawals from the contract in case of unforeseen circumstances. This is why the whenNotPaused modifier is attached to the transferTo but missed on the claim function.

Recommendation

add the whenNotPaused modifier on the claim function.

Informational Severity Issues

ERC20 import and inheritance is redundant

Resolved

Path

Node.sol

Description

ERC20.sol is getting imported and inherited even when the abstract ERC20Permit already inherits the ERC20. hence the erc20 import and inheritance is redundant and can be removed.

Recommendation

Redundant Import and inheritance can be removed.

Incorrect Claim Update Logic in transferTo() Function

Acknowledged

Path

Airdrop.sol

Function

claim()

Description

transferTo() function has a logical issue in the following line

@> claims[user] += amount;

Recommendation

@> claims[user] = amount;

Incorrect Event Emission

Resolved

Path

Airdrop.sol

Function

claim()

Description

ClaimSuccessful event is emitted incorrectly. Emit statement uses parameters as follows:
@>emit ClaimSuccessful(msg.sender, amount);

However, the declared event structure is:
@>event ClaimSuccessful(string email, address account);

This mismatch causes a compilation error since the data types and order do not align.

Recommendation

Change the event declaration:
event ClaimSuccessful(address account, uint256 amount);

Floating Solidity Version

Resolved

Path

Airdrop.sol/NODE.sol

Function

>=0.8.25

Description

All contracts under audit have a floating solidity version declared as >=0.8.25. This will cause the contracts to be deployed with any setup on the hardhat configuration.

Recommendation

Use a fixed version.

Remove unused imported library

Acknowledged

Path

IPool.sol

Function

"@openzeppelin/contracts/utils/structs/EnumerableSet.sol"

Description

This unused library imported will contribute to the byte code of the smart contract since it was imported yet it was not used in the Airdrop contract.

Recommendation

this library if not required for the contract should be removed.

Unpause function was omitted in the interface

Resolved

Path

IPool.sol

Function

unpause

Description

the IPool interface has all of the core functions but does not include the unpause functions. If this interface is set up to interact with the Airdrop contract, it would revert when the unpause function is invoked due to the absence of the function.

Recommendation

include this function in the interface to avoid possible revert.

Simplify the updateWhitelist to only perform whitelisting and non-whitelisting

Resolved

Path

NODE.sol

Function

updateWhitelist

Description

the IPool interface has all of the core functions but does not include the unpause functions. If this interface is set up to interact with the Airdrop contract, it would revert when the unpause function is invoked due to the absence of the function.

Recommendation

include this function in the interface to avoid possible revert.

Functional Tests

Some of the tests performed are mentioned below:

- initialize() function correctly sets _setRoleAdmin and _grantRole by utilizing AccessControlUpgradeable.
- DEFAULT_ADMIN_ROLE can mint at any time but only up to the MAX_SUPPLY limit.
- mint function allows tokens to be minted only by those with the MINTER_ROLE. However, it takes one day to call it, which means DEFAULT_ADMIN_ROLE can mint only one token and then run the function in the next 24 hours.
- Zero tokens are allowed to be burned in the burn function.
- Only the DEFAULT_ADMIN_ROLE can burn NODE tokens; no dummy tokens can be burned.
- initialize() function correctly sets roles.
- Deposit function work perfectly
- Zero amount is passed in TransferTo function but after it work correctly
- [PASS] test_If_TransferBatchFunctionHasWrongLengthValues()
- withdraw function works as expected

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 NodeOPS. We performed our audit according to the procedure described above.

Some issues of medium, low and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In the End, NodeOPS team resolved almost all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in NodeOPS. 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 NodeOPS. 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 NodeOPS 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

QuillAudits

6+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1K+** Projects Secured

Follow Our Journey

AUDIT REPORT

December, 2024

For

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillhash.com