QuillAudits

Audit Report
August, 2023

For

CARBIFY
CARBON NEUTRALITY. GAMIFIED.

Table of Content

Executive Summary 01

Checked Vulnerabilities 04

Techniques and Methods 05

Manual Testing 06

A. Common Issues 06

B. Contract - Aoc2 07

C. Contract - NFTree 08

Functional Tests 10

Automated Tests 10

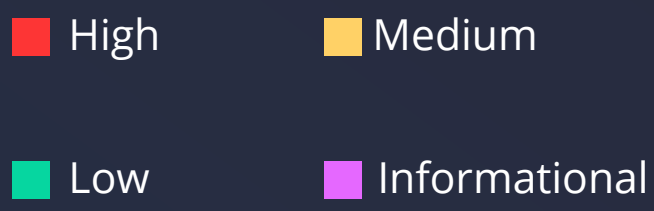
Closing Summary 11

About QuillAudits 12

Executive Summary

Project Name	Carbify
Project URL	*https://www.carbify.io/*
Overview	Aco2 is an ERC20 token that allows assigned addresses to either mint or burn tokens. It inherits the Openzeppelin Access Control Contract to achieve the role assignment. NFTree is an ERC721 upgradable contract inheriting Initializable,ERC721Upgradable,ERC721Enumerable, ERCBurnableUpgradable, AccessControlUpgradable from Openzeppelin library. All these aid in achieving the upgradability of the contract, the burning of tokens by assigned BURNER_ROLE, enumeration of minted tokens, and also for the designation of roles. The NFTree contract allows the MINTER to mint one or batch of tokens into an account and with an approval of the token owner, the BURNER can burn an ERC721 token.
Audit Scope	*https://github.com/Carbify-official/smart-contracts*
Contracts in Scope	Aco2 and NFTree
Commit Hash	31638cf6e94272066f8685d160dc22d450e5e056
Language	Solidity
Blockchain	Polygon and Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	16 August 2023 - 23 August 2023
Updated Code Received	23 August 2023
Review 2	24 August 2023
Fixed In	702730c1b969217b027ffcd9633c701cb3c4c945

Executive Summary



	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	0	0	3

Types of Severities

High

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

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

Low-level severity issues can cause minor impact and or 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 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.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour 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 analysed 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

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 analysed, 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 behaviour 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, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Common Issues

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

A.1 Unlocked pragma (pragma solidity ^0.8.17)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Resolved

A.2 Removed Unused Interfaces in Both Contracts

```
...
import "@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721EnumerableUpgradeable.sol";

...
import "./Interfaces/ITree.sol";
```

```
...
import "contracts/aco2/interfaces/ICRC.sol";
      Syed Uzair, 5 months ago • All contracts reworked …
```

Description

There are a few imports made in both contracts to be used but were not called within them.
In Aoc2 - ICRC.sol
In NFTree - IERC721EnumerableUpgradable and ITree

Remediation

Remove all unused interfaces in both contracts.

Status

Resolved

B. Contract - Aoc2

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

C.1 Additional Check for Zero Address Cost High Gas Consumption

```
require(to != address(0), "Invalid receiver");
```

Description

In the burn and mint functions, there were two checks to prevent that the to address and account addresses are not a null address and the second check is to prevent minting and burning of zero amount. While the second check is appropriate, the present of the first check will hike the cost of gas for calling these functions. The _mint and _burn functions from the ERC20 openzeppelin library already has this check to prevent minting or burning

Remediation

It is recommended to remove the first check that prevents the null address as inputs for both burn and mint functions in order to save gas.

Status

Resolved

C. Contract - NFTree

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

General Recommendation

In the NFTree contract, it allows the minter to batch mint as many tokens as possible but with a check present to prevent it does not mint beyond the initial supply indicated in the Batch struct. This is a good way to prevent running into an out of gas error when the minter passes a higher value greater than the batch initial supply. However, when the initial supply of the batch is still high, it will cause the out of gas error. It is recommended that the initial supply value of a batch be set to a value enough to cover for gas.

Functional Testing

Some of the tests performed are mentioned below:

- Should get the name of the token
- should get the symbol/tinker of the token
- should get the decimal of the token
- should get the total supply of the token after minted by the minter
- should get balance of the owner when contract is deployed
- should transfer tokens to other address
- should approve another account to spend token
- should mint and increase total supply
- Should deploy the NFTree and ensure it cannot be initialized more than once
- Should batch mint tokens to the provided address
- Should get all the tokens owned by an address
- Should revert when addresses without the MINTER and BURNER role call principal functions
- Should return the appropriate baseURI sets with the batch struct

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.

Summary

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

Some issues of informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

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

850+
Audits Completed

$30B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2023

For

CARBIFY
CARBON NEUTRALITY. GAMIFIED.

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com