QuillAudits

Audit Report
May, 2023

For

CARBIFY
CARBON NEUTRALITY. GAMIFIED.

Table of Content

Executive Summary

Project Name	Carbify
Timeline	12 May, 2023 - 16 May, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	Carbify Scope of work Consist of Two Contracts:- 1]Nftree.sol - It's extended from openzepplin's erc721enummerable with the extra features like loading/unloading cbys from the nft, batch minting nftree and it's foundation trees. 2]NftreeFactory.sol - It's a factory contract to clone the implementation of Nftree and its other foundation trees. It basically has a main function "AddBatch" that deploys a new batch of trees. The scope of this audit was to analyze the changes in Carbify smart contract's codebase for quality, security, and correctness. **Github**: *https://github.com/Carbify-official/smart-contracts* **Commit:** 6a822f6e59a8ec6f4d1becee665ba3f233c72b51
Fixed In	1a9869cf8b4d7d0e7c240dfd9100c629be34d003

High Medium

Low Informational

	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	1	2	0	1

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

High Severity Issues

1. Overlaping of tokenId indexes of NFTs would lead minting of NFT at same tokenId

Description

At NftreeFactory.sol#L151-152 incorrect firstIndex and lastIndex get calculated during ecoEmpire and backup initialization. ecoEmpire's batch firstIndex and lastIndex is shortfall by 1 while backup's batch firstIndex and lastIndex is shortfall by 2.

Because of this overlapping of indexes ecoEmpire would mint its NFT at the lastIndex of Staking NFT similarly backup NFT get minted at ecoEmpire's lastIndex.

Remediation

We recommend to replace the existing ecoEmpire firstIndex and lastIndex with the following (newBatch.lastIndex + 1) * 2 & (newBatch.lastIndex + 1) * 2 + totalSupply - 1) respectively. While for backup firstIndex and lastIndex with the following (newBatch.lastIndex + 1) * 3 & (newBatch.lastIndex + 1) * 3 + totalSupply - 1) respectively.

Status

Resolved

Medium Severity Issues

2. Incorrect fees precedence

Description

At Nftree.sol#L220-221, In updateFees function fee1, fee2 & fee3 are set but all are validated against MAX_FEE i.e. fee1, fee2, fee3 <= MAX_FEE , It is a valid check but the updateFees should have additional check where it validates that MAX_FEE >= fee1 > fee2 > fee3 otherwise it will defeat the purpose of having different fees value for different period of deposit. E.g. paying more fee for keeping the deposit of CBY for 2 years compare to the deposit of 6 months.

Remediation

We recommend to replace require statement at #L220 & #L221 with the following.

```
require(_fee2 < _fee1, "fee2 must be less than fee1");
```

```
require(_fee3 < _fee2, "fee3 must be less than fee2");
```

Status

Resolved

3. Multiple deposits of CBY for a tokenId would lead to pay more fees by the users.

Description

At Nftree.sol#L601, Deposit time is set as the block.timestamp, However, if the user deposits more tokens for a same tokenId after a year then user deposit timestamp get set as the latest deposit timestamp and user have to pay fee1 bracket even for the withdrawal of the early deposit amount. This would discourage the recurring deposits corresponds to a single tokenId.

Remediation

We recommend to use time weighted average instead of resetting with latest timestamp.

Status

Resolved

Low Severity Issues

No issues found

Informational Issues

4. Micelleanous recommendation for the code cleanup

Description

At Nftree.sol#L236 & Nftree.sol#L249, Recommended to not use the require statement in the view functions instead return false & (false, 0) respectively. It would be better for the UX.

At Nftree.sol#L253, This line can be simplified by rewriting as

```
return (timeBlacklisted !=0, timeBlacklisted)
```

At all depositCby variants function code checks whether the balanceOf and the allowance of msg.sender is sufficient, However it is redundant as ERC20's transferFrom does this during the execution.

Status

Resolved

Functional Testing

The complete functional testing report has been attached here.

Some of the tests performed are mentioned below

- ✓ should be able to setImplementation.
- ✓ should be able to add tree's in batch.
- ✓ should be able to set CBY address.
- ✓ should be able to set max batch mint.
- ✓ should be able to set public mint flag.
- ✓ should be able to set minting status.
- ✓ should be able to set carbify program.
- ✓ should be able to update fees.
- ✓ should be able to safe mint batch nft.
- ✓ should be able to safe mint batch nft to a given address.
- ✓ should be able to blacklistTree.
- ✓ should be able to depositCby.
- ✓ should be able to deposit CBY for multiple tokenIds.
- ✓ should be able to deposit CBY for all tokenIds owned by msg.sender.
- ✓ should be able to withdraw given CBY amount for a given tokenID.
- ✓ should be able to withdraw all CBY amount for a given tokenID.
- ✓ should be able to withdraw given CBY amount for multiple tokenIDs.
- ✓ should be able to withdraw all CBY amount for multiple tokenIDs.

Closing Summary

One issue of High severity was found, while two issue of medium severity & one issue of information severity were found which are now fixed by developers. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Carbify Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Carbify Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2023

For

CARBIFY
CARBON NEUTRALITY. GAMIFIED.

QuillAudits

Canada, India, Singapore, United Kingdom

www.quillaudits.com

audits@quillhash.com