QuillAudits

Audit Report
April, 2023

For

CARBIFY
CARBON NEUTRALITY. GAMIFIED.

Table of Content

Executive Summary

Project Name	Carbify
Overview	Carbify is an ERC20 token contract integrated with a buy and sell tax fees derived when users buy. The contract inherits the standard openzeppelin library for token creation. The contract owner can whitelist addresses to exclude from tax fees.
Timeline	11 April, 2023 to 14 April, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Carbify's CBY codebases for quality, security, and correctness. *https://github.com/Carbify-official/smart-contracts/blob/main/CBYV2.sol*
Commit hash	2bc7a4e3c2b48dfab213484521e93fafef650cd6
Fixed In	5241f8237ca6401e4460009b4a5485a72532dda6

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	1	1	4

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. The Implementation in the TransferFrom Function will Prevent Users from Spending All tokens in their Possession - *POC*

```
ftrace | funcSig
function transferFrom(address from↑, address to↑, uint256 amount↑) public override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from↑, spender, amount↑);
    uint256 fee;

    if(isRouter[_msgSender()] && isPair[to↑]) {
        fee = (sellFees * amount↑) / 10000;
        if(fee > 0) {
            require(balanceOf(from↑) >= amount↑ + fee, "ERC20: balance less than Amount (to be
            transfered) + DEX sell fee");
            _transfer(from↑, TREASURY_WALLET, fee);
        }
    }
    _transfer(from↑, to↑, amount↑);

    return true;
}
```

Description

The transferFrom function is designed to allow spenders use approved tokens and supposing the spender is the router and the receiver of the spent token is the pair address, accumulate the sell tax fee and transfer to the treasury wallet and afterwards transfer amount to the pair. This function expects users to approve routers with an extra amount to the pool. Say if a user intends to swap 3 ether(30000000000000000) worth of CBY tokens for eth, the user has to approve the router to spend 30600000000000000000. This will cause users to use less than what they approve of the router and still have some tokens in their possession.

Remediation

To allow users use all tokens in their possession, it is recommended that fee be deducted from the amount passed to this _transfer(from, to, amount).

Status

Resolved

Medium Severity Issues

2. Users Providing Liquidity Into Dex Pool Pays Taxes - *POC*

Description

The transfer and transferFrom function in the contract is designed to get allow for transfer of tokens and whenever it involves interacting with the DEX platform, it charges for taxes. This charges extends to the processes of adding liquidity to the pool.

Recommendation

It is recommended to design contract with a mapping that will exclude addresses that will provide tokens into the created pair in dexes.

Status

Resolved

Low Severity Issues

3. Owners Can Set Buy/Sell Tax to Arbitrary Values

```
ftrace | funcSig
function setBuyPercentage(uint256 _buy) external onlyOwner {
    buyFees = _buy;
}

ftrace | funcSig
function setSellPercentage(uint256 _sell) external onlyOwner {
    sellFees = _sell;
}
```

Description

The above functions give the contract owner the privileges to set values of buy and sell tax percentage. This could cause a rugpull if attackers overhaul the contract owner address.

```
// Equivalent to 2 percent. the last 2 0s is added to change the fee to 2 digit decimal in future if
required
uint256 public buyFees = 200; // ~ 2 percent
uint256 public sellFees = 200;
```

Recommendation

To prevent the owner from setting the tax fees to ridiculous tax that could cause a rugpull, it is recommended to make these state variables constant variables and remove the function that gives owners the privilege to set tax fee. It could be also designed to have a maxFee to prevent setting values way more than 2% if there will be anytime in the future to set the fee less than 2%.

Status

Resolved

Informational Issues

4. Floating Solidity Version

Description

Both contracts have a floating solidity pragma version. This is also present in inherited contracts. Locking the pragma helps to ensure that the contract does not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. The recent solidity pragma version also possesses its own unique bugs.

Recommendation

Making the contract use a stable solidity pragma version prevents bugs that could be ushered in by prospective versions. Using a fixed solidity pragma version while deploying is recommended to avoid deployment with versions that could expose the contract to attack.

Status

Resolved

5. Insufficient Code Comments

Description

The code comments currently are not sufficient Good code comments are relevant for easy comprehension of the contract; it aids users and developers interacting with the contract to duly understand the motive behind every function in the contract.

Recommendation

It is recommended to add comments to every existing function in the contract. The Natspec format is a well-detailed format that will aid in achieving good comment and documentation.

Status

Resolved

6. Unclear Error Message in Modifier

```
modifier onlyOwner() {
    require(_msgSender() == owner, "ERC20: Caller on Owner");
    _;
}
```

Description

Error message used for the require check in the onlyOwner modifier is not clear.

Recommendation

Give a clearer error message or use a custom error handler to save gas and aid clarity.

Status

Resolved

7. Missing Events for Critical State Changes

Description

Events emission are relevant to deployed contracts because it allows for filtering of emitted information. When certain state changes are happening, it is expected to emit these significant actions to track these changes.

- setRouterAddress
- setPairAddress

Recommendation

Consider emitting an event whenever certain significant changes are made in the contracts.

Status

Resolved

8. Missing Test Cases

Description

The codebase lacks unit test coverage. It is advisable to have test coverage greater than 95% of the codebase to reduce unexpected functionality and help fuzz test as many invariants as possible.

Recommendation

Include unit tests for the codebase.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below

- Should get the name of the token
- Should get the symbol of the token
- Should get the total supply of tokens minted to the contract owner
- Should make users provide liquidity into the pool.
- Should successfully transfer all tokens to the pair address when the buy tax is set
- Should transfer tokens to the DEX and increase tax collector balance
- Should confirm the treasury balance after a swap from exacts tokens to eth
- Should increase the balance of user when he swap from eth to CBY tokens
- Should allow spender transfer tokens to the DEX and increase tax collector balance
- Should allow DEX send to users and increase tax collector balance

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

Some issues of high, 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.

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
April, 2023

For

CARBIFY
CARBON NEUTRALITY. GAMIFIED.

QuillAudits

Canada, India, Singapore, United Kingdom

www.quillaudits.com

audits@quillhash.com