CleanCarbon

smart contracts
final audit report

April 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the CleanCarbon team to perform an audit of their smart contract. The audit was conducted between 09/04/2022 and 24/04/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code was provided via zip-file.

MD5 file sums are listed below:

CarboToken - 73afefcd9a3dc5679c6c7a137e56ae10

Configurator - 90395212b68dc0934315da3ce8486e8c

CrowdSale - d66e041b4f6f2f8ad340003c7061111b

DividendManager.sol - 0157aea3b981fd5151d5b99e2dc50faf

FeeHolder.sol - 81e68e6c50ab11f8ba98281d5614c858

FeeManager.sol - e7b7caf2922b637ede22aee1c5f700d2

RecoverableFunds.sol - 379a7e32f2c3907ac8a3ae82db60a8f5

VestingWallet.sol - 653e3f83695aab00fa44e64d3612a0d4

WithCallback.sol - 84e45a26c0a0f5326d3cac66b3e6b72c

ABDKMathQuad.sol - f7eb835f3f42668e5ff53b1d597fa3ae

Schedules.sol - 668b72a3d5dcdd3558a55bc90cba77f8

Stages.sol - 054052c6e7d2d9e9414f76a495a85c80

2.1 Summary

Project name	CleanCarbon
URL	https://cleancarbon.io/
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
CarboToken	
Configurator	
CrowdSale	
DividendManager	
FeeHolder	
FeeManager	
VestingWallet	
WithCallback	

RecoverableFunds

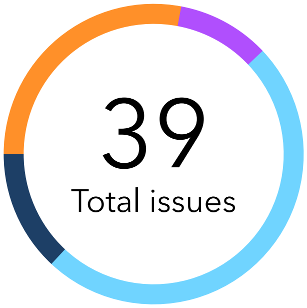
ABDKMathQuad

Schedules

Stages

Multiple contracts

3. Found issues



●	High	11 (28%)
●	Medium	4 (10%)
●	Low	19 (49%)
●	Info	5 (13%)

C1. CarboToken

ID	Severity	Title	Status
C1-01	● High	100% fees	? Open
C1-02	● High	excludeFromRFI() abuse	? Open
C1-03	● Medium	Blocked transfers	? Open
C1-04	● Medium	Expensive callback	? Open
C1-05	● Low	Floating Pragma	? Open
C1-06	● Low	Not enough events	? Open
C1-07	● Low	Gas optimization	? Open

C2. Configurator

ID	Severity	Title	Status
C2-01	Low	Floating Pragma	Open
C2-02	Low	Gas optimization	Open
C2-03	Info	Time dependent variables	Open

C3. CrowdSale

ID	Severity	Title	Status
C3-01	Low	Floating Pragma	Open
C3-02	Low	Not enough events	Open
C3-03	Low	Gas optimization	Open
C3-04	Low	No check for zero address	Open
C3-05	Low	Checks for price	Open
C3-06	Info	Typo	Open

C4. DividendManager

ID	Severity	Title	Status
C4-01	High	Owner overpower	Open
C4-02	High	Not enough requires	Open
C4-03	Low	Math rounding	Open

C5. FeeHolder

ID	Severity	Title	Status
C5-01	● Medium	Confusing onlyManager() modifier	Open
C5-02	● Low	Gas optimizations	Open

C6. FeeManager

ID	Severity	Title	Status
C6-01	● High	RFI support	Open
C6-02	● Low	Gas optimizations	Open
C6-03	● Low	Swaps without slippage and deadline control	Open

C7. VestingWallet

ID	Severity	Title	Status
C7-01	● High	Vesting schedule governance is centralized	Open
C7-02	● High	RFI support	Open
C7-03	● High	Owner can change token contract address	Open
C7-04	● High	Indexation problem	Open
C7-05	● High	Deposit index is not checked	Open
C7-06	● High	Owner access to the funds	Open

C7-07	● Medium	User's input check	Open
C7-08	● Low	No length info for schedules variable	Open
C7-09	● Low	Possible gas limit problem	Open

C8. WithCallback

ID	Severity	Title	Status
C8-01	● Info	Failed calls aren't logged	Open

C9. RecoverableFunds

ID	Severity	Title	Status
C9-01	● Low	Native token transfer	Open
C9-02	● Low	ERC20 transfers	Open

C13. Multiple contracts

ID	Severity	Title	Status
C13-01	● Low	Events for ownable functions	Open
C13-02	● Info	SafeMath usage	Open
C13-03	● Info	Floating pragma	Open

4. Contracts

C1. CarboToken

Overview

An RFI token, that has the ability to distribute commissions among token holders sending a part to the owner's addresses to replenish liquidity, buyback, and accumulate dividends. For examples of similar contracts, please follow the links below: SafeMoon, ReflectFinance.

Issues

C1-01 100% fees ● High ⓘ Open

The contract owner has the ability to set a fee of 100% for each type of fee. This may cause the users to receive nothing in an attempt to transfer (buy or sell) their tokens.

Also note, the total sum of fees may exceed 100%.

Recommendation

It is necessary to limit the amount of fees that the owner can set.

C1-02 excludeFromRFI() abuse ● High ⓘ Open

The owner of the token contract can redistribute portions of the tokens from users to a specific account. For this, an owner can exclude an account from the reward and include it back later. This redistributes part of the tokens from holders in profit of the included account. The abuse mechanism can be seen in **Appendix C**. In the provided attack test case the owner redistributes about 10% of other users' balance to the owner's balance.

Recommendation

We suggest restricting exclusion/inclusion methods by locking ownership for the maximum possible amount of time. As an alternative, you can rethink over the methodology for recalculating balances when calling the `includeInRFI()` function.

C1-03 Blocked transfers ● Medium ⓘ Open

All contract transfers can be blocked if the contract owner pauses the contract. In this case, users lose the ability to manage their tokens.

Recommendation

Users should be able to freely dispose of their tokens at any time.

C1-04 Expensive callback ● Medium ⓘ Open

Many of the contract functions have built-in callback functions. The contract owner has the ability to set any functionality for these callbacks. For example, a callback can perform expensive functions in other contracts. This can significantly increase the transaction cost for the user.

Recommendation

Restrict the owner's use of callback functions.

C1-05 Floating Pragma ● Low ⓘ Open

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.

Recommendation

Lock the pragma version and also consider known bugs (link) for the compiler version that is chosen.

C1-06 Not enough events ● Low Open

There is a lot of set functions in the code, but they don't have events.

Recommendation

Add events for the following functions: `setFees()`, `setFeeAddresses()`, `setTaxable()`, `setTaxExempt()`, `excludeFromRFI()`, `includeInRFI()`.

C1-07 Gas optimization ● Low Open

a. The functions `name()`, `symbol()`, `decimals()` can be declared as external to save gas.

b. The function `_transfer()` L65 should be called after checking the allowances inside the `transferFrom()` function (like in the `burnFrom()` function). This saves gas in case of a failure.

C2. Configurator

Overview

This contract is needed to initialize the initial values in the variables of other contracts.

Issues

C2-01 Floating Pragma ● Low Open

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.

Recommendation

Lock the pragma version and also consider known bugs (link) for the compiler version that is chosen.

C2-02 Gas optimization ● Low ⓘ Open

The variable `owner` of the `Amounts` structure is never used and can be removed.

C2-03 Time dependent variables ● Info ⓘ Open

At the time of the audit, the values of some variables are set in the past (L52, L56, L60, L64).

C3. CrowdSale

Overview

This contract is used for the primary sale of CarboTokens. The sale can be carried out using a white list.

Issues

C3-01 Floating Pragma ● Low ⓘ Open

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.

Recommendation

Lock the pragma version and also consider known bugs (link) for the compiler version that is chosen.

C3-02 Not enough events ● Low ⓘ Open

There are many 'set' functions in the code that don't emit events, which complicates the tracking of important off-chain changes.

Recommendation

Add events for the following functions: `setToken()`, `setVestingWallet()`, `setPercentRate()`, `setFundraisingWallet()`, `setPrice()`, `setStage()`, `addToWhitelist()`, `removeFromWhitelist()` and `receive()`.

C3-03 Gas optimization ● Low ⓘ Open

All `public` functions of the contract, except the `getActiveStageIndex()` and `calculateInvestmentAmounts()` functions, can be declared as `external` to save gas.

C3-04 No check for zero address ● Low ⓘ Open

The `setFundraisingWallet()`, `setToken()`, and `setVestingWallet()` functions do not check input values for zero address.

C3-05 Checks for price ● Low ⓘ Open

The final price of the tokens depends on two variables: `price` and `percentRate`. These variables can be set by the owner. We recommend using validation in the `setPrice()` and `setPercentRate()` functions to avoid errors when determining the price.

C3-06 Typo ● Info Open

There is a typo in the code documentation at L105 'purchasesd' should be replaced with 'purchased'.

C4. DividendManager

Overview

Distributor contract with calculations based on the user's balance of external token contract.

Issues

C4-01 Owner overpower ● High Open

The owner has the ablitiy to change the system tokens' addresses. Setting the wrong addresses may block the dividends withdrawals, setting malicious addresses would allow dividends balances manipulations.

```
function setToken(address _token) public onlyOwner {
    token = ICarboToken(_token);
}

function setBUSD(address _busd) public onlyOwner {
}
```

Recommendation

Remove these functions or restrict the access.

C4-02 Not enough requires ● High Open

In the `includeInDividends()` function there is no check for whether an account is excluded and in the function `excludeFromDividends()` there is no check for whether an account is included. This can crash the math of the contract.

Also there is no guarantee that this contract exludes and includes the same users as the CARBO token.

Recommendation

Consider including the safety checks.

C4-03 Math rounding ● Low ? Open

ABDK math library for floating point calculations may cause rounding errors in casting operations, e.g. `fromUInt()` and `toUint()`.

C5. FeeHolder

Overview

Simple contract holding any ERC20 tokens. Managed by limited number accounts.

Issues

C5-01 Confusing onlyManager() modifier ● Medium ? Open

`onlyManager()` modifier checks the `msg.sender` to be equal to the owner address, not the manager address.

```
modifier onlyManager() {
    require(owner() == _msgSender(), "LiquidityHolder: caller is not the manager");
    _;
}
```

Recommendation

Remove the manager variable or change the Error message in L15.

C5-02 Gas optimizations

● Low Open

Variables `manager` and `token` should be declared as immutables.

C6. FeeManager

Overview

Contract that gets the accumulated fees from FeeHolders, partially swaps them and distributes them to 4 addresses.

Issues

C6-01 RFI support

● High Open

The contract doesn't support tokens with commissions or rebase or RFI tokens. If this contract isn't excluded from fees for CARBO token, the math would crash.

Recommendation

Actual transfer amounts could be checked via balance before and after with the reentrancy possibility in mind.

C6-02 Gas optimizations

● Low Open

Variables `carbo`, `busd`, `uniswapRouter`, `buyFeeHolder`, and `sellFeeHolder` should be declared as immutables.

C6-03 Swaps without slippage and deadline control ● Low Open

`_swap()` function calls the Uniswap-like router without `amountOutMin` and `deadline` parameters. `swapAndDistribute()` transactions could be sandwiched.

C7. VestingWallet

Overview

Linear vesting contract based on Schedules library.

Issues

C7-01 Vesting schedule governance is centralized ● High Open

Owner is able to update schedules locking the user's funds, for example, in case of removing the schedule index with active deposits or extend the unlocking time for eternity.

```
function setVestingSchedule(uint256 id, uint256 start, uint256 duration, uint256 interval)
public override onlyOwner returns (bool) {
    return schedules.set(id, Schedules.Schedule(start, duration, interval));
}

function removeVestingSchedule(uint256 id) public onlyOwner returns (bool) {
    return schedules.remove(id);
}
```

Also, the owner can help users to withdraw their funds earlier than the end of the vesting.

Recommendation

We recommend saving the chosen vesting schedule for each user and each deposit so the owner is unable to control withdrawals.

C7-02 RFI support

● High Open

The contract doesn't support tokens with commissions or rebase or RFI tokens. If this contract isn't excluded from fees for CARBO token, the math would crash.

Recommendation

Actual transfer amounts could be checked via balance before and after with the reentrancy possibility in mind.

C7-03 Owner can change token contract address

● High Open

The `setToken()` function should not be callable after the first deposit, otherwise, user's funds would be locked.

```
function setToken(address tokenAddress) public override onlyOwner {
    token = IERC20(tokenAddress);
}
```

Recommendation

Remove the function or restrict its access.

C7-04 Indexation problem

● High Open

`withdraw()` function contains the `for()` loop over the schedules' length. There's no guarantee the EnumerableSet is ordered in natural numbers. Some active deposits may be excluded from the loop.

```
function withdraw() public returns (uint256) {
    uint256 tokens;
    for (uint256 index = 0; index < schedules.length(); index++) {
    Balance storage balance = balances[index][msg.sender];
    (...)
}
```

Recommendation

Use `schedules.at(index)` to retrieve key value.

C7-05 Deposit index is not checked ● High ? Open

Deposit is allowed for the wrong schedule index. In that case user's funds are locked.

```
    function deposit(uint256 schedule, address beneficiary, uint256 amount) public
override {
        token.transferFrom(msg.sender, address(this), amount);
        _deposit(schedule, beneficiary, amount);
    }

    function _deposit(uint256 schedule, address beneficiary, uint256 amount) internal {
        Balance storage balance = balances[schedule][beneficiary];
        balance.initial = balance.initial.add(amount);
        emit Deposit(schedule, beneficiary, amount);
    }
```

Recommendation

Check the `schedules.contains(schedule)` returning value.

C7-06 Owner access to the funds ● High ? Open

Owner has two ways how he can withdraw all users' funds:

1. Through functions of RecoverableFunds contract retrieve all contract's balance

2. Using `setBalance()` function, increasing his balance and then withdrawing the whole contract's balance

Recommendation

Deny the owner's access to user's funds.

C7-07 User's input check ● Medium ⓘ Open

In the `setVestingSchedule()` function there is no check for whether the argument `duration` is bigger than argument `interval`. If this property is violated then the user won't be able to withdraw part of their funds before the end of the vesting because function `_calculateVestedAmount()` will revert. Also if property `duration mod interval==0` is violated then the user can withdraw their funds earlier than the end of the vesting.

C7-08 No length info for schedules variable ● Low ⓘ Open

There is no function that returns the length of the `schedules` variable.

C7-09 Possible gas limit problem ● Low ⓘ Open

Unlimited `schedules` length may cause a gas limit exceedance during deposit or withdraw attempt. Owner must pay attention to the `schedules` set size.

C8. WithCallback

Overview

Simple contract that implements the callback of the CarboToken.

Issues

C8-01 Failed calls aren't logged ● Info ⓘ Open

`catch` sections of `try/catch` calls are empty.

C9. RecoverableFunds

Overview

Linear vesting contract, uses the Schedules library. No issues were found.

Issues

C9-01 Native token transfer ● Low ⓘ Open

It isn't recommended to use the `transfer()` function to transfer the native token. It is better to use the `call()` function.

C9-02 ERC20 transfers ● Low ⓘ Open

For some tokens the IERC20 interface can't be used for the transfer function and tx with this call will fail. It is better to use the SafeERC20 library from OpenZeppelin.

C10. ABDKMathQuad

Overview

Smart contract library of mathematical functions operating with IEEE 754 quadruple-precision binary floating-point numbers (quadruple precision numbers). Forked from @abdk-consulting/abdk-libraries-solidity repository. No issues were found.

C11. Schedules

Overview

Simple library of vesting schedules based on EnumerableSet from OpenZeppelin. No issues were found.

C12. Stages

Overview

Simple library of sale stages based on EnumerableSet from OpenZeppelin. No issues were found.

C13. Multiple contracts

Overview

Issues that are related to almost all reviewed contracts.

Issues

C13-01 Events for ownable functions ● Low ⓘ Open

Almost all owner-restricted functions haven't appropriate events.

C13-02 SafeMath usage ● Info ⓘ Open

In Solidity ^0.8.0 there is no need in using SafeMath because it is already embedded in language.

C13-03 Floating pragma ● Info ⓘ Open

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.

5. Conclusion

11 high and 14 medium severity issues were found.

The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix C. Hardhat framework test for possible abuse of excludeFromRFI()

```
const { expect } = require("chai");
const { formatUnits, parseEther } = ethers.utils;

describe("CarboToken token", function () {
  it("should run exclude include attack", async function () {
    this.timeout(120000);
    const [owner, alice, bob] = await ethers.getSigners();
    const CarboToken = await ethers.getContractFactory("CarboToken");
    const token = await CarboToken.deploy();

    const decimals = await token.decimals();
    const formatAmount = (amount) => formatUnits(amount, decimals);

    console.log("excluding owner from reward");
    await token.excludeFromRFI(owner.address);
    const totalSupply = await token.totalSupply();
    await token.transfer(alice.address, totalSupply.div(2));

    console.log(`total supply: ${formatAmount(totalSupply)}`);
    let balance = await token.balanceOf(owner.address);
    console.log(`owner balance is: ${formatAmount(balance)}`);

    await token.setTaxable(alice.address, true);
    await token.setTaxable(bob.address, true);
    await token.setFees(0, 100, 100, 100, 100, 100);

    const txCount = 120;
    console.log(`\nsending ${txCount} txAmount transactions between users`);
    const txAmount = "3000000000000000000000000";
    for (let i = 0; i < txCount; i++) {
      await token.connect(alice).transfer(bob.address, txAmount);
      const bobBalance = await token.balanceOf(bob.address);
      await token.connect(bob).transfer(alice.address, bobBalance);
    }

    balance = await token.balanceOf(owner.address);
    console.log(`owner balance is: ${formatAmount(balance)}`);
    const aliceBalance = await token.balanceOf(alice.address);
    console.log(`alice balance is: ${formatAmount(aliceBalance)}`);

    console.log("\nincluding address back to reward");
    await token.includeInRFI(owner.address);

    const newOwnerBalance = await token.balanceOf(owner.address);
    console.log(`owner balance is: ${formatAmount(newOwnerBalance)}`);

    const newAliceBalance = await token.balanceOf(alice.address);
    const aliceLoss = aliceBalance.sub(newAliceBalance);
    console.log(`alice balance is: ${formatAmount(newAliceBalance)}`);
    console.log(
      `alice loss is: ${aliceLoss
        .mul(100)
        .div(aliceBalance)}% or ${formatAmount(aliceLoss)} tokens`
    );
    const ownerProfit = newOwnerBalance.sub(balance);
    console.log(
      `owner profit is: ${ownerProfit.mul(100).div(balance)}% or ${formatAmount(
        ownerProfit
      )} tokens`
    );
  });
});
```

Test output

```
  CarboToken token
excluding owner from reward
total supply: 500000000.0
owner balance is: 250000000.0

sending 120 txAmount transactions between users
owner balance is: 250000000.0
alice balance is: 8656037.271673891883513669

including address back to reward
owner balance is: 276919558.838544438235594537
alice balance is: 7723970.453100055879649158
alice loss is: 10% or 932066.818573836003864511 tokens
owner profit is: 10% or 26919558.838544438235594537 tokens
     should run exclude include attack (9443ms)
```

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx

BLOCKCHAIN SECURITY