QuillAudits

Audit Report
April, 2022

For

The Climate Ecosystem

Table of Content

Executive Summary

Project Name	*Climate Ecosystem*
Overview	Blockchain project that can help save the world through the power of the many. Get returns on your investments while donating to Climate Change Organizations.
Timeline	March 29, 2022 - April 9, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Climate Ecosystem contract's codebase for quality, security, and correctness. Master
Source Code	*https://drive.google.com/drive/folders/1mK3cD-P2WGXsD2BbfIdle6omgbU7zZVP?usp=sharing*
Fixed In	*https://drive.google.com/drive/folders/1E6Ppny1th_iYe3EA9XF7SwiLY-PjA13E?usp=sharing*

High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	3	10
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	2	1	5

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. Voting Amplification Attack

Contract: CARBC.sol

Description

Votes can be amplified for a representative above the amount of tokens a delegator holds.

Exploit scenario

- A delegates 100 votes to B
- B now has 100 votes
- A transfers 100 tokens to C. _transfer function dont calls _moveDelegates and doesn't ensure the movement of votes from B to C. (assume that In this case transfer dont deducts fee from 100 tokens while transferring for simplicity)
- C now has 100 tokens, C delegates votes to B.
- Because the transfer function doesn't ensure the movement of delegated votes when transferred to someone, B's votes have been amplified to 200 votes.

Remediation

Add _moveDelegates() function in _transfer() function so movement of token transfers will ensure the movement of delegated votes.
Eg: _moveDelegates(_delegates[sender], _delegates[recipient], amount);

Status

Fixed

2. Voting Displacement Attack

Contract: CARBC.sol

Description

_transfer() doesn't ensure movement of votes from sender’s delegate/representative to the recipient's delegate/representative. In this case the contract is prone to voting displacement attack, Attacker can dismiss/decrease anyone’s vote using this attack.

Exploit Scenario

- B address has 500 votes delegated by some other addresses.
- Attacker1 has 1 token.
- Attacker1 delegates (through [#L256] _delegate) vote to B.
- B address holds 501 delegated votes now.
- Attacker2 has 500 tokens and he transfers 500 tokens to Attacker1. 2% fees get deducted so Attacker1 receives 490 tokens.
- Attacker1 now has 491 tokens.
- Attacker1 re-delegates his votes (through [#L256] _delegate) to Attacker2.
- [#L256] _delegate calls _moveDelegates, here old representatives' votes are decreased by amount which would be 491 in this case (That means 501 - 491) and new representative’s votes get increased by amount .
- Now B address has only 10 votes (501 - 491 = 10) and Attacker2 gets 491 votes.
- In above scenario B address loses 490 votes from 500 votes which it had in starting.

```
256     function _delegate(address delegator↑, address delegatee↑) internal {
257         address currentDelegate = _delegates[delegator↑];
258         uint256 delegatorBalance = balanceOf(delegator↑); // balance of underlying LCARBs (not s
259         _delegates[delegator↑] = delegatee↑;
260 
261         emit DelegateChanged(delegator↑, currentDelegate, delegatee↑);
262 
263         _moveDelegates(currentDelegate, delegatee↑, delegatorBalance);
264     }
265 
```

```
266     function _moveDelegates(
267         address srcRep↑,
268         address dstRep↑,
269         uint256 amount↑
270     ) internal {
271         if (srcRep↑ != dstRep↑ && amount↑ > 0) {
272             if (srcRep↑ != address(0)) {
273                 // decrease old representative
274                 uint32 srcRepNum = numCheckpoints[srcRep↑];
275                 uint256 srcRepOld = srcRepNum > 0
276                     ? checkpoints[srcRep↑][srcRepNum - 1].votes
277                     : 0;
278                 uint256 srcRepNew = srcRepOld.sub(amount↑);
279                 _writeCheckpoint(srcRep↑, srcRepNum, srcRepOld, srcRepNew);
280             }
281
282             if (dstRep↑ != address(0)) {
283                 // increase new representative
284                 uint32 dstRepNum = numCheckpoints[dstRep↑];
285                 uint256 dstRepOld = dstRepNum > 0
286                     ? checkpoints[dstRep↑][dstRepNum - 1].votes
287                     : 0;
288                 uint256 dstRepNew = dstRepOld.add(amount↑);
289                 _writeCheckpoint(dstRep↑, dstRepNum, dstRepOld, dstRepNew);
290             }
291         }
292     }
293
```

Recommendation

Add _moveDelegates() function in _transfer() function to ensure the movement of delegated votes when any token holder transfers any amount of tokens.
Eg: _moveDelegates(_delegates[sender], _delegates[recipient], amount);
Care needs to be taken here while moving delegated vote amount as some percent fees are getting deducted and getting sent to the donation address in the transfer function.

Status

Fixed

Medium Severity Issues

3. Excluded[] Length problem: Transaction may fail if the length of Excluded array is very large

Contract: Clime.sol

Description

Clime contract has two functions that use for loop over an array. includeInReward() and _getCurrentSupply(). If the length of the excluded array is very large, This may lead to extreme gas costs up to the block gas limit and eventually fail the transaction.

In an extreme situation with a large number of excluded addresses transaction gas may exceed the maximum block gas size and all transfers will be effectively blocked. If the owner's account gets compromised the attacker can make the token completely unusable for all users.

Exploit Scenario

Let's assume a scenario where the number of addresses excluded is very large and by running a for loop over that length, the transaction fails due to an out of gas issue. Now a user wants to transfer his token to another address. The user will call the transfer function, now if we look at the whole flow, eventually any one of the four transfer methods will be called and all those transfer functions will call _getValues(). And _getValues() will call _getCurrentSupply() which has a for loop which consumes a large quantity of gas. So if _getCurrentSupply() fails every transfer will fail.

```
function _getCurrentSupply() private view returns (uint256, uint256) {
    uint256 rSupply = _rTotal;
    uint256 tSupply = _tTotal;
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (
            _rOwned[_excluded[i]] > rSupply ||
            _tOwned[_excluded[i]] > tSupply
        )return (_rTotal, _tTotal);

        rSupply = rSupply.sub(_rOwned[_excluded[i]]);
        tSupply = tSupply.sub(_tOwned[_excluded[i]]);
    }
```

Recommendation

This issue can be avoided if the owner restricts the number of addresses excluded from the rewards. It is recommended to implement checks that prevent owner from adding addresses to exclude from rewards to a certain threshold.

Status

Acknowledged

4. Redelegation Failure

Contract: CARBC.sol

Description

Transaction may fail while re-delegating votes if the caller currently holds more tokens than what he was holding while delegating votes to the current delegate address.

Exploit Scenario

- A has 100 tokens.
- A delegates votes to B
- B now holds 100 votes.
- C sends a small amount of token, let's say 1 token to A.
- A wants to re-delegate votes to D.
- In this case A's current delegate holds 100 votes.
  while re-delegating A calls [#L256] _delegate which again calls [#L266] _moveDelegates.

  In [#L266] _moveDelegates, on [#L278] it will subtract the current balance amount of A (which is 101) from the vote amount of the current representative which is 100 , In this case transaction will revert because of integer underflow error.

```
266     function _moveDelegates(
267         address srcRep↑,
268         address dstRep↑,
269         uint256 amount↑
270     ) internal {
271         if (srcRep↑ != dstRep↑ && amount↑ > 0) {
272             if (srcRep↑ != address(0)) {
273                 // decrease old representative
274                 uint32 srcRepNum = numCheckpoints[srcRep↑];
275                 uint256 srcRepOld = srcRepNum > 0
276                     ? checkpoints[srcRep↑][srcRepNum - 1].votes
277                     : 0;
278                 uint256 srcRepNew = srcRepOld.sub(amount↑);
279                 _writeCheckpoint(srcRep↑, srcRepNum, srcRepOld, srcRepNew);
280             }
281
282             if (dstRep↑ != address(0)) {
283                 // increase new representative
284                 uint32 dstRepNum = numCheckpoints[dstRep↑];
285                 uint256 dstRepOld = dstRepNum > 0
286                     ? checkpoints[dstRep↑][dstRepNum - 1].votes
287                     : 0;
288                 uint256 dstRepNew = dstRepOld.add(amount↑);
289                 _writeCheckpoint(dstRep↑, dstRepNum, dstRepOld, dstRepNew);
290             }
291         }
292     }
```

Recommendation

Add _moveDelegates() function in _transfer() function to ensure the movement of delegated votes when any token holder transfers any amount of tokens.

Status

Fixed

5. Use of extcodesize can be exploited.

Contract: CARBC.sol

Description

On [#L77] _transfer() checks if the sender is a contract and if yes then it don't takes donation fee from the sender, for checking the address is contract or not [#L381] isContract() function uses extcodesize opcode which may be circumvented by a contract address during construction when it does not have source code available.

Remediation

We recommend reviewing logic and adding another mechanism for checking the contract address is EOA or contract. In the case if the address is known for which check is being made, use a conditional statement to check if the sender address is the one for which fee dont need to be deducted.

Status

Fixed

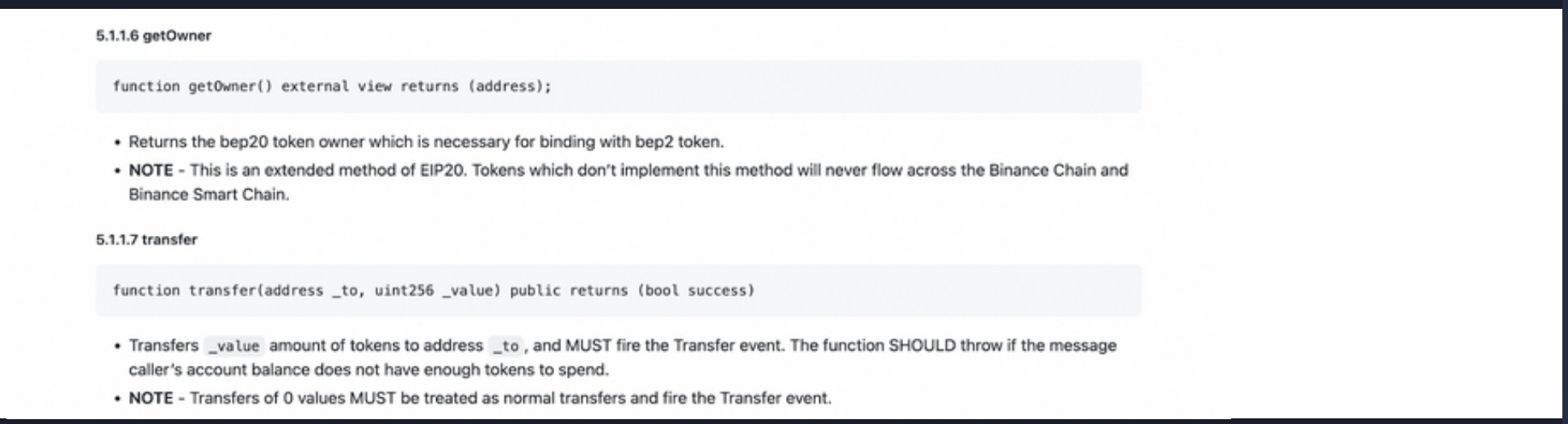
Low Severity Issues

6. BEP20 Standard violation

Contract: Clime.sol

Description

Implementation of transfer() function does not allow the input of zero amount as it's demanded in ERC20 and BEP20 standards. This issue may break the interaction with smart contracts that rely on full BEP20 support. Moreover, the GetOwner() function which is a mandatory function is missing from the contract.



5.1.1.6 getOwner

```
function getOwner() external view returns (address);
```

- Returns the bep20 token owner which is necessary for binding with bep2 token.
- **NOTE** - This is an extended method of EIP20. Tokens which don't implement this method will never flow across the Binance Chain and Binance Smart Chain.

5.1.1.7 transfer

```
function transfer(address _to, uint256 _value) public returns (bool success)
```

- Transfers `_value` amount of tokens to address `_to`, and MUST fire the Transfer event. The function SHOULD throw if the message caller's account balance does not have enough tokens to spend.
- **NOTE** - Transfers of 0 values MUST be treated as normal transfers and fire the Transfer event.

Recommendation

The transfer function must treat zero amount transfer as a normal transfer and an event must be emitted. Moreover, it is recommended to implement getOwner() function.

Reference

https://github.com/bnb-chain/BEPs/blob/master/BEP20.md#5117-transfer

Status

Acknowledged

7. Incorrect Error Message

Contract: Clime.sol

Description

includeInReward() function displays an incorrect error message if the require statement fails.

Remediation

The message "Account is already excluded" can be changed to "Account is not excluded" or “Account is already included”.

Status

Fixed

8. addLiquidity() recipient issues

Contract: Clime.sol

Description

addLiquidity() function in CLIME.sol calls for uniswapV2Router.addLiquidityETH() function with the parameter of lp tokens recipient set to owner address. With time the owner address may accumulate a significant amount of LP tokens which may be dangerous for token economics if an owner acts maliciously or its account gets compromised. This issue can be fixed by changing the recipient address to the Clime contract or by renouncing ownership which will effectively lock the generated LP tokens.

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) public {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
```

Recommendation

Add _moveDelegates() function in _transfer() function to ensure the movement of delegated votes when any token holder transfers any amount of tokens.

Status

Acknowledged

9. The owner can abuse ExcludeFromReward() functionality and prevent users from earning rewards.

Contract: CARBC.sol

Description

As per the flow of the contract, whenever a token transfer is made, all the token holders receive a share of the transaction fees. The owner of the token contract can redistribute part of the tokens from users to a specific account. For this owner can exclude an account from the reward and include it back later. This will redistribute part of the tokens from holders in profit of the included account and the excluded account will not receive the reward.

Exploit scenario

The Owner can temporarily exclude an account from earning a reward and include it back later after some time.

Remediation

We suggest lock exclusion/inclusion methods by locking ownership for the maximum possible amount of time. Moreover, excluding/including an account from earning rewards is a very sensitive functionality that involves the interference of certain privileged users (owner). Whenever any significant action is performed by the privileged users, an event must be emitted.

Status

Acknowledged

Informational Issues

10. Unlocked Pragma (^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested 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 by removing the caret to lock the file onto a recent Solidity version.

Status

Acknowledged

11. Missing zero check

Description

Contracts lack zero address checks, hence are prone to be initialized with zero addresses.

In CARBC.sol:
- constructor()

In CLIME.sol:
- constructor()
- excludeFromFee()
- includeInFee()
- inCaseTokensGetStuck()

Recommendation

Consider adding zero address checks in order to avoid risks of incorrect contract initializations.

Status

Acknowledged

12. Missing Testcases

Description

Test cases for the code and functions have not been provided.

Recommendation

It is recommended to write testcases of all the functions. Any existing tests that fail must be resolved. Tests will help in determining if the code is working in the expected way. Unit tests, functional tests, and integration tests should have been performed to achieve good test coverage across the entire codebase.

Status

Acknowledged

13. Public functions that could be declared external inorder to save gas

Description

Whenever a function is not called internally, it is recommended to define them as external instead of public in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function's parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.

Here is a list of function that could be declared external:

- totalSupply()
- name()
- symbol()
- decimals()
- increaseAllowance()
- decreaseAllowance()
- isExcludedFromReward()
- setNumTokensSellToAddToLiquidity()

Status

Acknowledged

14. Variables That Could Be Declared As Constant inorder to save gas.

Contract: Clime.sol

Description

There are multiple variables in the contract, whose value is never updated, it is recommended to declare those variables as constant.

Here is a list of variable that could be declared constant:

- Name
- Symbol
- Decimals
- _tTotal

Status

Acknowledged

15. Unindexed event parameters

Description

MinTokensBeforeSwapUpdated, SwapAndLiquify events don't have any indexed parameters. Unindexed parameters make it difficult to track important data for off-chain monitoring tools. Moreover, in SwapAndLiquify there is a typo in 3rd parameter tokensIntoLiqudity, it should be tokensIntoLiqiudity.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
event SwapAndLiquifyEnabledUpdated(bool enabled);
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiqudity
);
```

Recommendation

Consider indexing event parameters to avoid the task of off-chain services searching and filtering for specific events.

Status

Acknowledged

16. Third Party Dependencies

Description

The logic of the contract requires it to interact with third-party protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of the Clime contract requires interaction with third-party protocols. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Status

Acknowledged

17. Incorrect Naming Convention

Description

The _withdrawBNB() function in clime.sol is an external function. However, the name of the function has an underscore before it. This type of convention is mostly used to define private function. Since this is an external function, it is recommended to remove the underscore from the name of the function.

Status

Fixed

18. Gas optimization: For loop optimization

Description

In includeInReward() and _getCurrentSupply(), there is a for loop which iterates the value of _excluded.length times. Each time the for loop executes _excluded.length is calculated which consumes some gas. This can be optimized by calculating the value of _excluded.length outside the for loop. The optimized loop would look like:

```
function includeInReward(address account) external onlyOwner {
    require(_isExcluded[account], "Account is already excluded");
    uint NoOfExcludedAddress =  _excluded.length;
    for (uint256 i = 0; i < NoOfExcludedAddress; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[NoOfExcludedAddress - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Recommendation

Update the loop as recommended above.

Status

Fixed

19. Redundant Code

Description

The token _tokenTransfer() function in Clime.sol, has an if-else block and in the second else if block, there is _transferStandard(sender, recipient, amount). This condition is already satisfied by the else block hence it is safe to remove the extra code from the contract,

```
function _tokenTransfer(
    address sender,
    address recipient,
    uint256 amount,
    bool takeFee
) private {
    if (!takeFee) removeAllFee();

    if (_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferFromExcluded(sender, recipient, amount);
    } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
        _transferToExcluded(sender, recipient, amount);
    } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferStandard(sender, recipient, amount);
    } else if (_isExcluded[sender] && _isExcluded[recipient]) {
        _transferBothExcluded(sender, recipient, amount);
    } else {
        _transferStandard(sender, recipient, amount);
    }
```

Recommendation

Remove the Redundant code from the contract.

Status

Acknowledged

20. Use of block.timestamp for trade deadline

Description

Deadline set as block.timestamp, In this case of delay too less time margin for the transaction to execute, may add the risk of the transaction being reverted by router contract because of the expired deadline.
Too big deadline/time margin may add risk of miner manipulation, where a miner can hold transaction and can execute or add it to block at profitable time.

```
553     function swapTokensForEth(uint256 tokenAmount1) private {
554         console.log("swapTokensForEth");
555         // generate the uniswap pair path of token -> weth
556         address[] memory path = new address[](2);
557         path[0] = address(this);
558         path[1] = uniswapV2Router.WETH();
559 
560         _approve(address(this), address(uniswapV2Router), tokenAmount1);
561 
562         // make the swap
563         uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
564             tokenAmount1,
565             0, // accept any amount of ETH
566             path,
567             address(this),
568             block.timestamp
569         );
570     }
571 
```

```
572     function addLiquidity(uint256 tokenAmount1, uint256 ethAmount1) private {
573         console.log("addLiquidity",tokenAmount1);
574         // approve token transfer to cover all possible scenarios
575         _approve(address(this), address(uniswapV2Router), tokenAmount1);
576 
577         // add the liquidity
578         uniswapV2Router.addLiquidityETH{value: ethAmount1}( // refunds ETH back
579             address(this),
580             tokenAmount1,
581             0, // slippage is unavoidable
582             0, // slippage is unavoidable
583             owner(),
584             block.timestamp
585         );
586         console.log("End addLiquidity");
587     }
```

Recommendation

Consider adding block.timestamp + some amount of seconds while adding a deadline.

Status

Fixed

21. Minimum amount to receive is 0

Description

Minimum amount of output tokens that must be received is 0, which allows trade to execute even when the output amount is very low.

```
553     function swapTokensForEth(uint256 tokenAmount) private {
554         console.log("swapTokensForEth");
555         // generate the uniswap pair path of token -> weth
556         address[] memory path = new address[](2);
557         path[0] = address(this);
558         path[1] = uniswapV2Router.WETH();
559 
560         _approve(address(this), address(uniswapV2Router), tokenAmount);
561 
562         // make the swap
563         uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
564             tokenAmount,
565             0, // accept any amount of ETH
566             path,
567             address(this),
568             block.timestamp
569         );
570     }
```

Recommendation

Consider adding a minimum amount to receive greater than zero, The minimum amount to receive may vary according to The token amount passed in while swapping for ETH.

Status

Acknowledged

22. Incorrect event argument

Description

[#L363] updateDonationRate() emits [#L370] DonationRateUpdated event, its using donationRate as argument for previousRate parameter. But on [#L369] entered _donationRate is getting assigned to donationRate. Because of this, the event will emit the same values for previousRate and newRate parameters of DonationRateUpdated event.

```
363     function updateDonationRate(uint16 _donationRate↑) public onlyOperator {
364         require(
365             _donationRate↑ < 10000,
366             "LCARB::updateDonationRate: Donation rate must not exceed the maximum rate."
367         );
368
369         donationRate = _donationRate↑;
370         emit DonationRateUpdated(msg.sender, donationRate, _donationRate↑);
371     }
372
```

Recommendation

Store value of previous donation rate in local variable before assigning it to donationRate and use that local variable as previousRate parameter while emitting event

Status

Fixed

23. Redundant modifier

Description

onlyOwner modifier can be used instead of defining and using new onlyOperator modifier.

Recommendation

Consider removing onlyOperator modifier.

Status

Acknowledged

24. Owner can mint more tokens than maximum supply cap

Description

[#L50] mint() function allows minting of tokens more than maximum token supply cap of 21000000.

Recommendation

Consider adding require statement/ conditional check to revert transaction while minting to keep amount less than or equal to max supply limit.

Status

Fixed

Functional Testing

Contract - Clime.sol

- Should test all the getter values
- Should test approve, allowance and transferFrom
- Should test increaseallownace and decrease allowance
- Should correctly calculate reflections from tokens and vice versa
- Should test transferStandard, transferFromexcluded, transferToExcluded, transferBothExcluded.
- Should test donation fees
- should test liquidity fees
- should check transaction fees.
- Fees must not be charged from excluded address.
- Excluded from rewards must not receive rewards.
- Should withdraw stuck tokens and BNBs.
- OnlyOwner modified must work as intended and must revert is called by non owner addresses.

Contract - CARBC.sol

- Should be able to mint tokens only by owner
- Should deduct donation fee if address is not excluded from fee
- Should be able to delegate votes to any address
- Should be able to delegate votes by signature
- Should be able to exclude from fee
- Should be able to include to fee
- Should be able to update donation rate
- Reverts if one signature used more than one time while delegating
- Reverts while minting if total token supply exceeds max supply limit.
- Should move votes to recipient's representative when tokens transferred

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

Several issues of High, Medium, and Low severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

At the end, some issues were resolved by the Auditee, while others were acknowledged by the Auditee.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Climate Ecosystem 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 Climate Ecosystem 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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2022

For

The Climate Ecosystem

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com