byterocket

Smart Contract Audit Report

CO2ken Tokenizer

8th of October 2021

Contents

1. Preface

The team of **Toucan Labs** contracted byterocket to conduct a smart contract audit of their CO2ken tokenizer smart contracts. Toucan Labs are building a "*protocol that allows anybody to bring carbon credits on-chain via tokenization, directly sell or aggregate them in carbon index pools*".
The team of byterocket started the review and audit of the smart contract on the 14th of September and finished the preliminary part of the audit on September 17th. The final audit started on October 4th and finished on October 8th.

The audit included the following services:

- *Manual Multi-Pass Code Review*
- *Automated Code Review*
- *In-Depth Protocol Analysis*
- *Deploy & Test on our Testnet*
- *Formal Report*

byterocket gained access to the code via their private GitHub repository. We started the audit on the master branch's state on September 14th, 2021.
Throughout our audit, changes have been made to the code, which we incorporated up until the latest state of the master branch on October 8th, 2021 6:30pm UTC (*commit hash ed604639ac85f27e60921a4326d2e19007d37d8f*).

The developers have responded to our findings in section (2.1.) via commit *e6819bde06dbfd3c5eb0b4af11d1b651b70d793b*, fixing all of the bugs or vulnerabilities we found throughout our audit.

2. Manual Code Review

We conducted a manual multi-pass code review of the smart contracts mentioned in section (1). Three different auditors went through the smart contract independently and compared their results in multiple concluding discussions.

These contracts are written according to the latest standards used within the Ethereum community and the Solidity community's best practices. The naming of variables is very logical and understandable, which results in the contract being easy to understand. The code is very well documented and up to the latest standards. We were very satisfied with the overall quality of the code.

On the code level, we **found no bugs or flaws**. Three previously found bugs of medium severity have promptly been fixed correctly by the developers, as noted in the updated list below. Besides that, we have noted some very minor other findings. A further check with multiple automated reviewing tools (*MythX*, *Slither*, *Manticore*, *and different fuzzing tools*) **did not find any additional bugs** besides some common false positives.

2.1 Bugs & Vulnerabilities

2.1.A - CarbonProjects.sol - Line 172 - 176 ~~[MEDIUM SEVERITY]~~ **FIXED**

```
string memory oldProjectId = projects[tokenId].projectId;
projectIds[oldProjectId] == false;

projects[tokenId].projectId = newProjectId;
projectIds[newProjectId] == true;
```

The lines 173 and 176 are not setting the values in the `projectIds` array, since they are using == instead of =, which leads to a comparison returning `true` or `false`. This does not set the variable to true or false, but instead does nothing. As this is probably not wanted, we advise the team to change this.

Fixed via commit b99b9e23e45f0edd97156e94aa0760250f031530.

2.1.B - BaseCarbonTonne.sol - Line 373 ~~[MEDIUM SEVERITY]~~ FIXED

```
function redeemSingle(
     address account,
     address erc20,
     uint256 amount
     ) public virtual whenNotPaused {
  require(
    tokenBalances[erc20] >= amount,
    'Cannot redeem more than is stored in contract'
  );
  _burn(account, amount);
  tokenBalances[erc20] -= amount;
  IERC20(erc20).safeTransfer(account, amount);
}
```

This function has no access control and thus allows anyone to execute it as long as the contract is not paused. It allows the calling user to provide a different `account` address, which is not their own. This would allow the calling user to redeem their BCT and retrieve the underlying ERC20. The user of the account address can not prevent this. On the other side, the redeemMany function in line 358 automatically inserts `msg.sender` instead of giving the calling user a choice.

We are not sure whether this behaviour is desired.

Update: The function was updated to check whether the `account` equals to `msg.sender` and was changed to be `internal` only.
Fixed via commit e6819bde06dbfd3c5eb0b4af11d1b651b70d793b.

2.1.C - Pool.sol - Line 137 ~~[MEDIUM SEVERITY]~~ FIXED

```
function redeemSingle(
     address account,
     address erc20,
     uint256 amount
     ) public virtual whenNotPaused {
  require(
    tokenBalances[erc20] >= amount,
    'Cannot redeem more than is stored in contract'
  );
  _burn(account, amount);
  IERC20(erc20).safeTransfer(account, amount);
}
```

This function has no access control and thus allows anyone to execute it as long as the contract is not paused. It allows the calling user to provide a different `account` address, which is not their own. This would allow the calling user to redeem their BCT and retrieve the underlying ERC20. The user of the account address can not prevent this. On the

other side, the redeemMany function in line 121 automatically inserts `msg.sender` instead of giving the calling user a choice.

We are not sure whether this behaviour is desired.

Update: The contract is being discontinued. Since the same bug was fixed in 2.1.B. we consider this one to be fixed as well.
Fixed via commit e6819bde06dbfd3c5eb0b4af11d1b651b70d793b.

2.1.D - BaseCarbonTonne.sol - Line 146-147 ~~[MEDIUM SEVERITY]~~ FIXED

```
function redeemSingle(
     address account,
     address erc20,
     uint256 amount
     ) public virtual whenNotPaused {
  require(
    tokenBalances[erc20] >= amount,
    'Cannot redeem more than is stored in contract'
  );
  _burn(account, amount);
  IERC20(erc20).safeTransfer(account, amount);
}
```

The deposit function in line 71 increases `tokenBalances` as well as `overallAmount` by the deposited amount. The redeem function, however, does not change these values even though a burn takes place.

We are not sure whether this behaviour is desired, especially since `tokenBalances` is explicitly checked in line 143.

Update: This has been correctly fixed through adding the corresponding calculation `tokenBalances[erc20] -= amount` to the code in line 147.
Fixed via commit e6819bde06dbfd3c5eb0b4af11d1b651b70d793b.

2.2 Other Findings

2.2.A - General [~~NO SEVERITY~~] FIXED

```
bool hasExternalFilter;
```

Some variables like the one above from `Pool.sol` do not have a visibility set. Solidity reverts to `internal/private` visibility if nothing is set, but we advise projects to set an explicit visibility to prevent problems in the future. Since most variables have a certain visibility set, we assume it has been forgotten here.

Fixed via commit 54d4bf752aaa97af5812a241f80703a12b57117e.

2.2.B - BaseCarbonTonne.sol - Line 128 - 134 [NO SEVERITY]

```
for (uint256 i = 0; i < standardsLen; i++) {
  if (addToList == true) {
    standards[_standards[i]] = true;
  } else {
    standards[_standards[i]] = false;
  }
}
```

Since the new value of `standards` depends entirely on `addToList`, it could also be simplified to

```
for (uint256 i = 0; i < standardsLen; i++) {
    standards[_standards[i]] = addToList;
}
```

Even though this does not save any gas (*<0.01%*) during execution as the compiler does the simplification anyways, it reduces the size of the contract, potentially slightly decreasing the deployment cost.

We leave it entirely up to the developers whether they want to change this or not.

2.2.C - BaseCarbonTonne.sol - Line 294-301 [NO SEVERITY]

```
if (internalWhiteList[erc20Addr]) {
  return true;
}

require(
  internalBlackList[erc20Addr] == false,
  'Error: TCO2 token contract blacklisted'
);
```

It stood out to us that the blacklist is handled after the whitelist, which immediately returns `true`. It might be the case that this is intended to work like this, but the usual pattern would be to check the blacklist first and only proceed to the whitelist if it's not listed there.

We just want to bring this to the attention of the developers in case they are not aware. We leave it entirely up to the developers whether they want to change this or not.

3. Protocol/Logic Review

Part of our audits are also analyses of the protocol and its logic. A team of three auditors went through the implementation and documentation of the implemented protocol.

We went through all of the provided documentation, tests and contracts in a very detailed manner. The general description of the protocol is very well made, it's very easy to understand how each function is supposed to work and what it implements.

The developers were not only very responsive but also provided videos and further documentation to us to help us understand each part of the protocol.

Besides the found bugs we have stated in section 2, we have **not been able to discover any problems** in the protocol itself implemented in the smart contracts.

We have deployed the contracts on an internal test net as well, where we try to force the contract into malicious states through randomized and predetermined inputs in a fuzzing manner. We were **not able to maliciously interact with the contract**.

4. Summary

During our preliminary code review (*which was done manually and automated*), **we found no bugs or flaws**. Three previously found bugs of medium severity have promptly been fixed correctly by the developers, as noted in the updated list in section (2.1.). Besides that, we have noted some very minor other findings. A further check with multiple automated reviewing tools **did not find any additional bugs** besides some common false positives.

Our brief protocol review and analysis did neither uncover any game-theoretical nature problems nor any other functions prone to abuse. According to our analysis, the protocol has been correctly implemented in the smart contracts as the documentation describes it. We were **not able to find any issues**.

In general, we are **happy** with the overall quality of the code, its tests and documentation. We did not find anything that leads us to believe that bigger problems are still to be uncovered. Overall, the developers of Toucan Labs did a very good job.