Smart Contract Audit Report

Toucan Protocol

NCT

24th of February 2022

Contents

1. Preface **3**

2. Manual Code Review **4**

2.1 Severity Categories 4

2.2 Summary 5

2.3 Findings 6

3. Protocol/Logic Review **10**

4. Summary **11**

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held understanding of the auditor's knowledge of security patterns as they relate to the client's contract(s), assuming that blockchain technologies, in particular, will continue to undergo frequent and ongoing development and therefore introduce unknown technical risks and flaws. The scope of the audit presented here is limited to the issues identified in the preliminary section and discussed in more detail in subsequent sections. The audit report does not address or provide opinions on any security aspects of the Solidity compiler, the tools used in the development of the contracts or the blockchain technologies themselves, or any issues not specifically addressed in this audit report.

The audit report makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, the legal framework for the business model, or any other statements about the suitability of the contracts for a particular purpose, or their bug-free status.

To the full extent permissible by applicable law, the auditors disclaim all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. The auditors hereby disclaim, and each client or user of this audit report hereby waives, releases and holds all auditors harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

1. Preface

The developers of **Toucan Protocol** contracted byterocket to conduct a smart contract audit of their **N**ature **C**arbon **T**onne (NCT) smart contract. Toucan is building a protocol that "*brings programmable carbon to Web3, unlocking its potential for a regenerative economy.*"

The team of byterocket reviewed and audited the above smart contracts in the course of this audit. We started on the 21st of January and finished on the 24th of February 2022.

The audit included the following services:

- *Manual Multi-Pass Code Review*
- *Protocol/Logic Analysis*
- *Automated Code Review*
- *Formal Report*

byterocket gained access to the code via a public GitHub repository. We based the audit on the master branch's state on February 11th, 2022 (*commit hash db8c8b3d218d89eac1b4e0dbc6dff0fe1210bf76*). The updated version was provided to us on March 18th, 2022 via new commits to the public repository, addressing our findings. The last and most recent commit hash that we audited is *2a537b34fdc3345ae68efcdefd316f707c4c2066*.

2. Manual Code Review

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

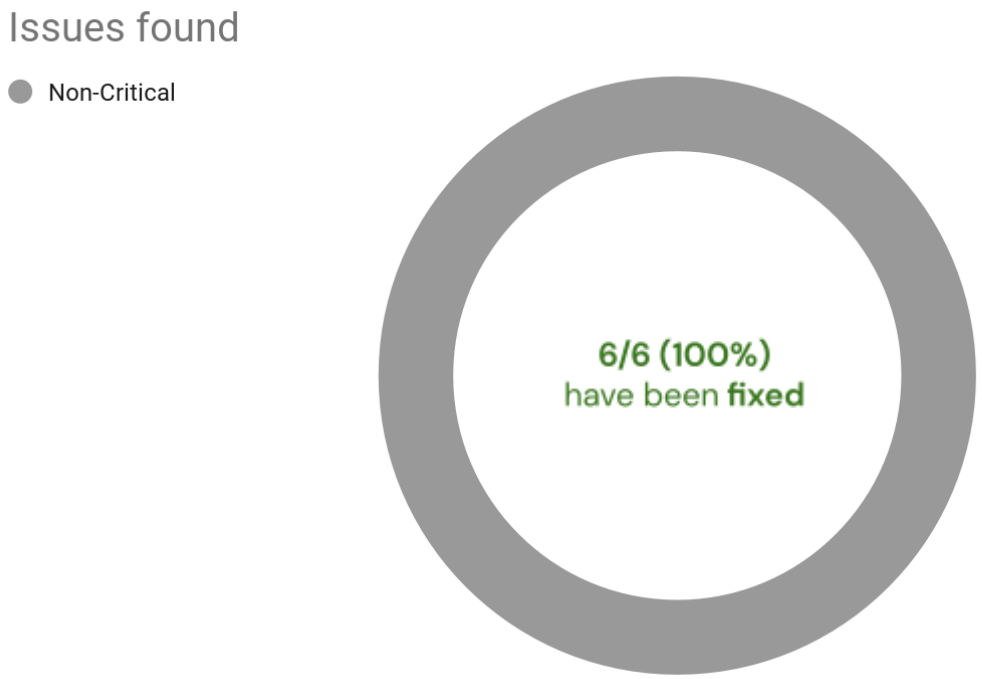
The manual review and analysis were additionally supported by multiple automated reviewing tools, like Slither, GasGauge, Manticore, and different fuzzing tools.

2.1 Severity Categories

We are categorizing our findings into four different levels of severity:

Non-Critical	Does not impose immediate risk but is relevant to security best practices. Includes issues with - Code style and clarity - Versioning - Off-chain monitoring
Low Severity	Imposes relatively small risks or could impose risks in the long-term but without assets being at risk in the current implementation. Includes issues with - State handling - Functions being incorrect as to specification - Faulty documentation or in-code comments
Medium Severity	Imposes risks on the function or availability of the protocol or imposes financial risk by leaking value from the protocol if external requirements are met.
High Severity	Imposes catastrophic risk for users and/or the protocol. Includes issues that could result in - Assets being stolen/lost/compromised - Contracts being rendered useless - Contracts being gained control of

2.2 Summary



On the code level, we **found 6 bugs or flaws, with 6 of them being fixed** in a subsequent update**.** Prior to this, there have been 6 non-critical findings.

The contracts are written according to the latest standard used within the Ethereum community and the Solidity community's best practices, with some exceptions listed below. The naming of variables is very logical and understandable, which results in the contract being useful to understand. The code is well documented. The developers provided us with a test suite as well as deployment scripts.

2.3 Findings

[FIXED] ~~[NON-CRITICAL]~~ NC.1 - Inaccurate documentation for setTCO2-Scoring()

Location: NCT.sol - Line 328 - 331

Description:
The documentation of this function has a slight inaccuracy. It can only be called by the manager role, while the documentation states that only owners may call it.

These are the affected lines of code:

```
/// @notice Allows owner to pass an array to hold TCO2 contract
addresses that are
/// ordered by some form of scoring mechanism
/// @param tco2s array of ordered TCO2 addresses
function setTCO2Scoring(address[] calldata tco2s) external onlyManagers
```

Recommendation:
Consider updating the documentation to match the implementation.

Update on the 29th of March:
The corresponding documentation has been updated.

[FIXED] ~~[NON-CRITICAL]~~ NC.2 - Use of deprecated library functions

Location: NCT.sol - Line 75 & BaseCarbonTonne.sol - Line 82

Description:
The `initialize()` function is using the `_setupRole()` function to set up roles. This function has been deprecated, as referenced here. The new function is called `_grantRole()`.

These are the affected lines of code:

```
_setupRole(DEFAULT_ADMIN_ROLE, sender);
```

Recommendation:
While `_setupRole()` internally just calls `_grantRole()`, it just makes sense to directly call `_grantRole()`.

Update on the 29th of March:
This has been changed; the new function is now being used.

[FIXED] ~~[NON-CRITICAL]~~ NC.3 - No events for state-changing functions

Location: Throughout NCT.sol & BaseCarbonTonne.sol

Description:
There are some functions (as described below) that change important state variables like supply caps or contract registries, which do not emit an event. Usually, it is the best practice to emit an event here.

These are the affected functions:

```
setToucanContractRegistry()
switchMapping()
setSupplyCap()
setMinimumVintageStartTime()
setTCO2Scoring()
```

Recommendation:
Consider adding a corresponding event to each of these functions.

Update on the 29th of March:
For each of our five recommended functions that didn't emit an event previously, there is now an event being emitted.

[ACKNOWLEDGED] ~~[NON-CRITICAL]~~ NC.4 - Missing check on time input

Location: NCT.sol - Line 318 - 326

Description:
The `setMinimumVintageStartTime()` function sets a quite important `timestamp` without checking the input. If this `timestamp` would be set in the future, deposits would be disabled for `ToucanContracts` that are not whitelisted. It is usually the best practice to validate important inputs.

These are the affected lines of code:

```
function setMinimumVintageStartTime(uint64 _minimumVintageStartTime)
         public virtual onlyOwner {
  minimumVintageStartTime = _minimumVintageStartTime;
}
```

Recommendation:
Consider adding a `require` statement that ensures, that invalid (e.g. future) `timestamps` can not be set.

Update on the 29th of March:
The team has acknowledged this finding. Due to a well-defined process from their side when calling these functions, they can be ensured, that this value will not be set incorrectly. The contract size tradeoff is a good reason to not add this check here.

[FIXED] ~~[NON-CRITICAL]~~ NC.5 - Internal function that could be public

Location: NCT.sol - Line 318 - 326

Description:
The `checkEligible` function returns whether a certain token is eligible to be deposited. This technically sounds like a function that some users might benefit from having publicly available before they deposit a certain token. It did not seem very clear why this function has been implemented as `internal`.

These are the affected lines of code:

```
function checkEligible(address erc20Addr) internal view virtual [...]
```

Recommendation:
Consider setting functions to `public` that might provide a benefit for some users if the function does not have to be `internal`.

Update on the 29th of March:
The function visibility has been set to `public`.

[ACKNOWLEDGED] ~~[NON-CRITICAL]~~ NC.6 - Possible reentrancy in deposit function

Location: NCT.sol - Line 344 - 367

Description:
Due to the positioning of the `safeTransferFrom` call in the `deposit` function, it is technically possible to execute a reentrancy attack for certain malicious tokens that are implemented in a certain way.

These are the affected lines of code:

```
IERC20Upgradeable(erc20Addr).safeTransferFrom(
  msg.sender,
```

```
  address(this),
  amount
);
```

Recommendation:
The chances of this happening are minimal since this is more of a theoretical threat. We mainly raised this issue to emphasize the fact that the team has to be very diligent when it comes to adding other tokens.

Update on the 29th of March:
The team acknowledged our finding and ensured that there is a diligent process in place that verifies any potential tokens that are being used.

3. Protocol/Logic Review

Part of our audits are also analyses of the protocol and its logic. The byterocket team went through the implementation and documentation of the implemented protocol.

The repository itself contained tests and documentation. We found the provided unit tests that are coming with the repository execute without any issues and cover the most important parts of the protocol.

According to our analysis, the protocol and logic are working as intended.

We were **not able to discover any additional problems** in the protocol implemented in the smart contract.

4. Summary

During our code review (*which was done manually and automated*), we **found 6 bugs or flaws, with 6 of them being fixed** in a subsequent update**.** Prior to this, there have been 6 non-critical findings. Our automated systems and review tools did **not find any additional ones**.

The protocol review and analysis did neither uncover any game-theoretical nature problems nor any other functions prone to abuse.

In general, there are some improvements that can be made, but we are **very happy** with the overall quality of the code and its documentation. The developers have been very responsive and were able to answer any questions that we had.