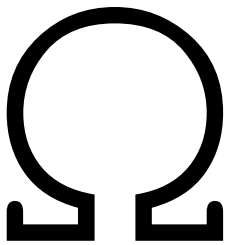
Toucan Bridge

Final Audit Report

July 11, 2022



Team Omega

teamomega.eth.limo

Summary

Toucan Protocol has asked Team Omega to audit the contracts that define the behavior of the bridge contracts.

We found **no high severity issues** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **one** issue as “medium” - these are issues we believe you should definitely address. In addition, **7** issues were classified as “low”, and **4** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Severity	**Number of issues**	**Number of resolved issues**
High	0	0
Medium	1	1
Low	7	7
Info	5	1

Scope of the Audit

We audited code from the following repositor:

```
https://github.com/ToucanProtocol/tokenizer/
```

And specifically the following Solidity contracts:

```
contracts/cross-chain/ToucanCrosschainMessenger.sol
```

```
contracts/cross-chain/ToucanCrosschainMessengerStorage.sol
```

Together with bridge-related changes in the files:

```
contracts/pools/BaseCarbonTonne.sol
```

```
contracts/pools/NCT.sol
```

The first audit report was based on the following commit:

```
fa768fb33f2d3fab8e52f03770b1ae93f878d35e
```

Resolution

The Toucan Protocol developers have subsequently addressed the findings in this report in the following commit:

```
c1d89a2a7e9655e9fab7b0be7a48f6256309c043
```

The Toucan developers have resolved all the issues we mentioned, except a number of issues marked as “info”, which do not present a security risk.

We have audited these changes and marked the resolution below

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues, concerning mostly the variables and functions visibility, were found and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

Centralization risks

The owner of the ToucanCrossChainMessenger contract has wide-ranging power to disrupt the system by its ability to mint any amount of new tokens or upgrade the contract. Specifically, the owner of the Toucan messenger contact can:

- call `upgradeTo` or `upgradeToAndCall` and change the logic of the messenger contract.
- call `renounceOwnership` and `transferOwnership` and transfer ownership to any account.
- call `addTokenPair` to map local tokens to remote tokens.
- can call `addTokenPair` and change an existing mapping (within one week after the last change).
- call `pause` and `unpause` and halt all operations on the bridge.
- call `enrollRemoteRouter` to register the address of a router (i.e. an accepted origin of cross chain messages)
- call `setAbacusConnectionManager` and set or change the `abacusConnectionManager` that controls the inboxes. Inboxes are addresses that can call `handle` and mint any amount of tokens.

The Abacus bridge has, of course, the possibility to mint any amount of tokens. These powers are managed by the Abacus connection manager (which is set by the owner of the messenger contract). The Abacus Connection manager itself has an owner role, and the owner account controls many crucial aspects of the connection manager - for example, the owner can register new “Inboxes”, which are addresses that can mint any amount of tokens.

In addition, the owner of the BCT and NCT contracts, which already had the power to upgrade those contracts, can, with the changes we audited, also call `setRouter` and give any address the power to mint any amount of tokens.

This makes for three separate admin-controlled roles that could, if compromised, do extreme damage to the value of the token by minting any amount of tokens. A compromise of these accounts would be categorized as “High Severity” on our severity scale.

Specific care should be taken by making sure these roles are only assigned to trusted accounts. Especially, the various “owner” roles should probably be held by multi-sig contracts with a non-trivial amount of signers (e.g. 3-out-of-5 or 4-out-of-7).

In addition, we strongly recommend that crucial functions (such as the `upgradeTo`, `transferOwnership`, `setRouter`), be called via a Timelock, so that if the multisig is compromised, users and the community have time to react and can possibly mitigate the damage.

General

G1. A floating pragma is set instead of a fixed pragma [info] [not resolved]

The Solidity pragma version used in the contracts is set as floating:

```
pragma solidity ^0.8.0
```

Setting it to a fixed version will make the compilation step more deterministic, and make it easier for third parties to verify the deployed bytecode.
Recommendation: Remove the `^` symbol in the pragma definition to make the solidity version fixed and ensure bytecode consistency.
Severity: Info
Resolution: This issue was not resolved. A range is now specified instead of a floating pragma, but this still leaves nearly the same level of ambiguity with regards to the compiler version used.

G2. Use the latest solidity version [info] [resolved]

The contracts use Solidity version 0.8.4, while the latest version is 0.8.14.
Recommendation: Use the latest Soldity version.
Severity: Info
Resolution: This issue was resolved as recommended.

G3. Invalid SPDX license identifier [info] [not resolved]

The contracts contain the following `SPDX-License-Identifier` invocation:

```
// SPDX-License-Identifier: UNLICENSED
```

“UNLICENSED” is not a known license identifier (cf. https://spdx.org/licenses/), and therefore the license expression is not valid
Recommendation: If the objective is to NOT release the software under any license at all, this purpose can be obtained by simply removing the SPDX license identifier.
Severity: Info. Although UNLICENSED is not a valid SPDX expression, it is often used in Solidity projects.
Resolution: This issue was not resolved. The developers communicated that they prefer to keep UNLICENSED as a placeholder until they decide on a proper license.

G4. Pin versions of solidity dependencies [info] [not resolved]

In `package.json`, a possible range of versions of the OpenZeppelin dependencies is specified rather than a single one:

```
"@openzeppelin/contracts": "^4.6.0",
"@openzeppelin/contracts-upgradeable": "^4.6.0",
```

This can lead to unexpected problems when a new version of OpenZeppelin is released and other developers (or the continuous integration process, or yourself at a later date) will recompile the contracts with this new version.
Recommendation: Specify fixed versions of smart contract dependencies in `package.json`, instead of ranges, so that the solidity code can be verified and there is no ambiguity about the actual code you are to deploy or already have deployed.
Severity: Info
Resolution: This issue was not resolved.

ToucanCrossChainMessenger.sol

T1. revertBridgeRequest allows the owner of the contract to mint an arbitrary amount of tokens [medium] [resolved]

The function `revertBridgeRequest` allows the owner of the ToucanCrossChainMessenger to mint new BCT or NCT tokens on the home chain for each historical bridging request.
This has different security implications, some of which we discussed with the team in private. We believe the main vulnerability here is that the owner effectively has a license to mint an unlimited amount of tokens, by repeating the following steps:

1. Send N tokens from the home chain to the destination chain, which will burn N tokens on the home chain and mint N new tokens on destination chain
2. Call `revertBridgeRequest` tokens and mint N tokens on the home chain
3. repeat

Recommendation: The intended use of this function is to give the owner of the messenger contract a way to recover when the bridge fails to deliver a message.
The function can be used for this purpose, although in that case we do recommend to implement a mechanism that will guarantee that the bridge does not deliver the message after

`revertBridgeRequest` was called (this could be done by implementing a function that invalidates incoming requests and calling that on the destination chain).
The power given here to the owner, namely to mint any amount of tokens when the bridge is functioning correctly, far exceeds the specific scenario of reverting the sending of an undelivered message.
In addition, there are many reasons that the bridge can fail; failure to deliver messages is just one of them, and relatively limited in its consequences.
We recommend to re-evaluate the failure modes of the bridge and their probability, and decide on the basis of that whether it is worth changing the security model of the bridge.
Severity: Medium. The architecture already has a considerable centralization risk (see the "privileged roles" section about), this functionality adds further risks by giving an administrative account (which is responsible to add new token pairs) unlimited minting power.
Resolution: This issue was resolved - the `revertBridgeRequest` was removed. Nonetheless, see the "Centralization risks" section for a more detailed analysis of other owner related risks.

T2. Cooldown period is unenforceable as it serves to limit the admin but the admin himself controls it [low] [resolved]

The function `revertBridgeRequest` is subject to a cooldown period - requests can not be reverted before at least `cooldownPeriod` seconds have passed.
This cooldown period is however not enforceable, as the owner can just set it to 0 and then immediately revert the request.
Recommendation: Remove the cooldown period, or, if it is needed, hardcode a value or make it settable by another agent that is not the owner.
Severity: Low
Resolution: This issue was resolved as recommended. The cooldown period (and the entire revert logic) was removed.

T3. Remove redundant import [low] [resolved]

The ToucanCrossChainMessenger file contains the following import statement:

```
import {XAppConnectionClient} from './optics/xapp-contracts/XAppConnectionClient.sol';
```

This import is not used and could be safely removed.
Recommendation: Remove the import statement.
Severity: Low
Resolution: This issue was resolved as recommended.

T4. Use a global nonce to save some gas [low] [resolved]

The messenger contract keeps a nonce for each address that sends messages to the bridge.

```
mapping(address => uint256) public nonce;
...
nonce[msg.sender] = currentNonce;
```

There is however no need to keep a separate nonce for each sender, and some gas can be saved by using a global nonce.
Recommendation: Use a simple state variable instead of a mapping for keeping track of the nonces

```
uint public nonce;
...
nonce = currentNonce;
```

Severity: Low
Resolution: This issue was resolved as recommended.

T5. Avoid using assembly [low] [resolved]

On line 50ff:

```
assembly {
  chainId := chainid()
}
```

These lines can be replaced by:

```
block.chainid
```

Recommendation: it is best practice to avoid using assembly if that does not lead to immediate gains, and we recommend to follow that practice here as well.
Severity: Low
Resolution: This issue was resolved as recommended.

T6. Unused function parameter in _handle function [info] [resolved]

The compiler emits the following warning:

```
Warning: Unused function parameter. Remove or comment out the variable
name to silence this warning.
   --> contracts/cross-chain/ToucanCrosschainMessenger.sol:121:9:
    |
121 |         bytes32 _sender,
    |         ^^^^^^^^^^^^^^^
```

The warning refers to the `_sender` argument in the `_handle` function - this is the address of the Router contract on the source chain.
Recommendation: Fix the warning, either by commenting out the variable name, or by using the variable, for example by adding the sender as an argument to the `BridgeRequestReceived` event.
Resolution: This issue was resolved as recommended. The variable name was commented out to remove the linter warning.

ToucanCrossChainMessengerStorage

S1. add a __gap variable to reserve storage space for upgrades [low] [resolved]

If the `ToucanCrossChainMessenger` contract will be deployed as an upgradeable proxy, we recommend preparing for future upgrades of the contract and reserve some space for any additional variables that may be defined in this class at a future date.
For more information, see the OpenZeppelin documentation
https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
Recommendation: Add a line such as the following to the contract, below the declarations of the other state variables:

```
uint256[50] private __GAP; // gap for upgrade safety
```

Severity: Low
Resolution: This issue was resolved. A `__gap` variable was not added, but all state variables were moved over from the `ToucanCrossChainMessenger` to the `ToucanCrossChainMessengerStorage` contract, so that the Storage contract controls the last slots in the storage layout of the contract. No extra gap is necessary with this structure. In future iterations, developers should take care that if they define state variables in the messenger contracts, it would be convenient to add a `__gap` to the storage contract.

NCT.sol and BaseCarbonTonne.sol

NB1. Make setRouter external [low] [resolved]

On line 684, the `setRouter` function can be declared `external`
Recommendation: Declare the function as external.
Severity: Low
Resolution: This issue was resolved as recommended.

NB2. allowance in bridgeBurn is not necessary [low] [resolved]

The bridgeBurn function requires that the account whose tokens are being burnt has given an explicit approval to burn these tokens:

```
    function bridgeBurn(address _account, uint256 _amount) external onlyRouter
{
        require(
            allowance(_account, msg.sender) >= _amount,
            'Not enough allowance'
        );
        _burn(_account, _amount);
    }
```

This function is only callable by the router, which is expected to be an instance of `CrossChainMessenger`. This means that the bridgeBurn function will be called as part of a sendMessage request, and that the value for _account passed to bridgeBurn is actually the msg.sender of the original request. In other words, we can be sure that the transaction has been initiated by the owner of the tokens that are being burnt, and so adding this check does not in any way add additional security.
Recommendation: Remove the check for allowance, as it does not add to security, but does present a UX annoyance for the message sender, which must sign and send two transactions instead of a single one.
Severity: Low
Resolution: The issue was resolved as recommended.

NB3. User allowance in bridgeBurn is not updated [low] [resolved]

The `bridgeBurn` function checks that the caller has an allowance to spend tokens. However, the allowance is not updated after the tokens have been burnt. This means that even if an account has set a limited allowance, the sender of the message the option can burn any amount of tokens by repeatedly calling the bridgeBurn function

Recommendation: Update the user allowance when tokens are burnt (unless of course you remove the check for allowance altogether as we recommend in NB2).

Severity: Low

Resolution: This issue was resolved by removing the allowance check altogether.

NB4. Duplicate functions - it is easier to inherit from a common source [info] [not resolved]

The `BaseCarbonTonne` and the `NCT` contracts each define a number of functions that are specifically used for managing the token bridge, such as `setRouter`, `onlyRouter`, `bridgeMint` and `bridgeBurn`. This means that about 40 lines of code are duplicated over two files.

Recommendation: we recommend refactoring the code: create a new base contract where these bridge-specific functions are defined, and inherit from that contract.

Severity: Info

Resolution: This issue was not resolved.