SOLIDIFIED

Audit Report for Aztec - June 17, 2022

Summary

Audit Report prepared by Solidified covering the Aztec protocol Ethereum Bridge contract for Compound Bridge.

The following report covers the **Compound Bridge**.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 17 June 2022.

Audited Files

The source code has been supplied in the form of one public Github repository.

https://github.com/aztecProtocol/aztec-connect-bridges/

Commit Hash: `ad5d8d5fa83ae0e1c519e1bdb79adb4caa5aa8c1`

```
src
|-- bridges
|   -- compound
      |-- CompoundBridge.sol
      |-- interfaces
```

Intended Behavior

Smart contract responsible for depositing, managing and redeeming Defi interactions with the Compound protocol.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than in a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	Medium	-
Test Coverage	High	-

Issues Found

Issue #	Description	Severity	Status
1	The safeIncreaseAllowance() might revert if the current allowance is not 0	Minor	Resolved
2	Only allow valid cToken contracts from Compound	Note/Precaution	-

Critical Issues

No issues found

Major Issues

No issues found

Minor Issues

1. The `safeIncreaseAllowance()` might revert if the current allowance is not `0`

For example Tether's `ERC-20` implementation does not allow modifying the existing allowance if the existing allowance is not equal to `0`. If allowance is not `0`, it first needs to be set to `0` and then it can be set to another value.
The `SafeERC20.safeIncreaseAllowance()` does not handle such a case, although the comments in code seem to assume otherwise.

See discussion at `https://github.com/OpenZeppelin/openzeppelin-contracts/pull/3046` for more details.

Currently, the RollupProcessor/cToken always takes the entire amount, however the `approve` would revert if that changes.

Recommendation
In order for the bridge to be compatible with underlying tokens which do not fully comply with the `ERC-20` specification, consider setting allowance to `0` before setting it to another value (it could be implemented for all `ERC-20` tokens or for a manageable specific list of tokens).

Resolved:
The issue has been fixed by the Aztec team. The approval for the `cToken` can be further simplified by just always approving the maximum uint amount instead of using the difference between the maximum and the allowance.

```
        if (allowance < type(uint256).max) {
              _cToken.approve(ROLLUP_PROCESSOR, type(uint256).max - allowance);
         }
```

Change to `_cToken.approve(ROLLUP_PROCESSOR, type(uint256).max);`

Informational Notes/Precaution

---

2. Only allow valid cToken contracts from Compound

Currently, the bridge doesn't verify if a passed cToken address is actually a valid cToken contract from Compound.

A potential malicious contract could steal funds, etc.

The `Comptroller` could be called to verify if the cToken is a valid market.

`Comptroller.markets(address cToken) public returns(bool)`

Disclaimer