Audit Report for Global Money - December 19, 2021

Summary

Audit Report prepared by Solidified covering the Global Money smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 19 December 2021.

A second audit on an extended scope was concluded on 28 November 2021.

Audited Files

The source code has been supplied in the form of two GitHub

repositories:https://github.com/Globalmoney/core

Commit number: `0584106b2e58fb452e574394dc1ee2f4714c1376`

https://github.com/Globalmoney/farm

Commit number: `bebdecb1a7a8b07b07218caad4bd7569ac99eaf4`

The scope of the audit was limited to the following files:

```
contracts
├── IDepositable.sol
├── GlobalMoney.sol
├── SingleLevelRateFeeder.sol
├── StableRateFeeder.sol
├── SwaplessConversionPool.sol
└── UpdatableStableRateFeeder.sol


contracts
└── PrivateFarming.sol
```

Intended Behavior

The smart contract implements a Binance-based protocol that interacts with the Pancakeswap bridge to store BEP-20 tokens (stablecoins) into the Binance blockchain.

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 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

Solidified found that the Global Money contracts contain no critical or major issues, 2minor issues, in addition to 4 informational notes.

We recommend all issues are amended, while the notes are up to the team’s discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	SwaplessConversionPool.sol: Missing Zero-checks	Minor	Resolved
2	Global Money.sol: Potential reentrancy with sometokens	Minor	Resolved
3	Pragma allows for a wide range of compiler versions	Note	Acknowledged
4	SwaplessConversionPool.sol: Unused argument in deposit function	Note	Acknowledged
5	SwaplessConversionPool.sol: Lack of sanity checks in migrate() function	Note	Resolved
6	uint to int casting	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. `SwaplessConversionPool.sol`: Missing Zero-checks

Some functions are not protected with pre-condition checks for 0 values.

`setOperationRouter`
`setExchangeRateFeeder`

Recommendation

Whilst this might be intentional in some cases, we recommend adding zero checks where a 0 value is undesired.

Status

Resolved

2. `GlobalMoney.sol`: Potential reentrancy with some tokens

The `deposit()` function may suffer from reentrancy if the protocol is used with malicious tokens or with Binance-777 tokens that provide a hook.
In addition, `finalizeWithdrawUpToUser()` also suffers from this and could become unexecutable.

Recommendation

Consider protecting these functions with a `ReentrancyGuard`.

Status

Resolved

Informational Notes

3. Pragma allows for a wide range of compiler versions

The `pragma` statement allows for a wide range of compiler versions, including some versions with known bugs. In addition, the language syntax has changed since the earlier versions that are allowed.
Additionally, restricting the `pragma` to using version 0.8.0 or larger would make using the SafeMath library unnecessary.

Recommendation
Consider limiting the compiler to at least a single major version number.

4. `SwaplessConversionPool.sol:` Unused argument in deposit function

The argument `_minAmountOut` is not used at all in the `deposit(uint256 _amount, uint256 _minAmountOut)` function.

Recommendation
Consider removing unnecessary arguments.

5. `SwaplessConversionPool.sol:` Lack of sanity checks in migrate() function

The function `migrate()` could be called with the wrong arguments during an upgrade, as it has no sanity checks that the target address is a contract of the right form.

Recommendation
Consider using an interfaceId check for the contract address, or calling a function that is known must exist in the target contract. Alternatively, a zero-guard should be added to this function too.

Status
Resolved

6. uint to int casting

The smart contracts try to convert `uint256` values to `int256` in multiple methods. It's harmless in most cases, but for large enough values, the values can be different from `uint256` resulting in an incorrect result.

Recommendation

Consider keeping the `uint256` values as it is.

Status

Resolved

Disclaimer