SOLIDIFIED

Audit Report for Bitgo November 19th, 2018.

Summary

Audit Report prepared by Solidified for Bitgo and Kyber covering the Wrapped Bitcoin smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on November 19th, 2018, and the final results are presented here.

Audited Files

The following contracts were covered during the audit:

- Controller.sol
- ControllerInterface.sol
- Factory.sol
- Members.sol
- MembersInterface.sol
- WBTC.sol
- IndexedMapping.sol
- OwnableContract.sol
- OwnableContractOwner.sol

Notes

The audit was based on the repository `https://github.com/WrappedBTC/bitcoin-token-smart-contracts/tree/audit2` `commit` hash `099ac7be60205d04a342773ab4ee0a5d58dd1f3b`, solidity compiler `0.4.24+commit.e67f0147`

Intended Behavior

Wrapped BTC (WBTC) is an ERC20 compliant token, pegged to Bitcoin. The system is composed of merchants and a custodian. Merchants have to request minting of tokens to the custodian, informing the BTC transaction hash, and the custodian (who controls the BTC private keys), then authorizes the minting request.

Burn requests follow a similar process process, where burns are executed by the merchant and acknowledged by the custodian, who will document the BTC transaction hash to provide auditability.
The basic assertion of the system is that the `totalSupply` of WBTC should not exceed the balances of all custodian's BTC addresses.

Trust points of the smart contracts are a DAO (not in scope for this audit), Merchants and Custodians. The DAO owns the controller and can upgrade parts of the dapp (the Factory.sol and the Member.sol can be upgraded by linking new contracts) and also manages access of merchants and custodians. Merchants and custodians are trusted to report/check existing BTC transactions before minting, with the custodian reviewing the validity of merchant requests. All transaction information is public and can be verified by anyone with access to both Ethereum and Bitcoin transaction data.

Issues Found

1. RenounceOwnership is not in accordance with specifications

The function `renounceOwnership`, inherited from OpenZeppelin's `Ownable.sol` in all contracts, could render the token frozen for minting/burning, and the controller contract no longer upgradeable.

While there are use cases for renouncing ownership of a contract, the in-scope contracts do not specify such a situation, resulting in unnecessary increase of the attack surface.

Recommendation
Consider overloading the `renounceOwnership` and disabling it, or at least requiring a transfer (waiting to be claimed) before the renouncement is executed.

Amended [22-11-2018]
The issue was fixed and is no longer present in pull request `Audit2 #1`. The pull request is yet to be merged.

2. Upgrading logic for Members.sol and Factory.sol may lead to inconsistent state, and does not allow for re-use of state / data

---

The specs determine that both Controller and Members can be upgraded, and in fact they can. The Controller contract relies on `WBTC.sol`, `Factory.sol` and `Members.sol` for state. Members.sol, however, stores the whole member list, and upgrading it without losing the data (or performing data migration) is not possible.

Similarly the Factory contract stores data regarding wallets (merchant and custodian). Upgrading one of the two, will cause inconsistencies between states that will demand manual data migration.

Recommendation
Consider making the `IndexedMapping.sol` contract an eternal storage. This would allow for upgrading member management logic while retaining previous state, while still allows for the complete purging of state if that is necessary.

Also consider storing all information on the same contract, ensuring consistency between merchant/controller addresses and BTC wallets associated with them.

3. Merchant can generate a large amount of small burn requests, increasing the operational cost (BTC fees) of the custodian

Merchants could attack the custodian, by dividing burn requests they receive into smaller transactions. This would cause the operational cost of the custodian to rise, along with the number of transactions sent to the BTC network.

Recommendation
Consider requiring a minimum burn value per transaction, or batching relevant transactions into a larger overall transaction (thereby decreasing operational cost).

4. Usage of strings for storing BTC transaction hashes and addresses is inefficient

Operations using Strings in Solidity, as other dynamically sized types, consume a considerably more gas than fixed sized types.

Recommendation
Consider using a fixed bytes type, such as bytes32, for storing BTC transaction and address information as this will enable considerable gas savings for all dapp actors, especially over a long period of usage.

5. Consider upgrading to recent versions

Solidity compiler 0.4.24 and Openzepellin-solidity 1.12 were used in the in-scope smart contracts.

Solidity 0.4.25 fixed a bug about the ** operator (not applicable to the in scope contracts), as well as some of the findings in the Solidity 0.4.24 audit. OpenZeppelin, version 2.0 provides granular access control (departing from the owner pattern), a stable API (enabling easier future upgrades to logic) and has been once again audited. It also includes updates to Claimable.sol and `CanReclaimToken.sol`, with an updated list of references to ERC20 token functionality.

Recommendation
Consider upgrading to the latest stable compiler and OpenZeppelin versions.

6. Static analysis denotes potential integer underflow

Static analysis warns of a potential integer overflow within the remove function in IndexedMapping.sol, resulting from an arithmetic operation (subtraction) in which the potential subtraction could result in a number below zero.

Traditionally, impacted code should include a `require` statement, mitigating concern regarding potential underflow. Similarly, subtraction can be followed by an `assert` statement, ensuring the two prior integers summed have not underflown (validating that the result is less than the two integers that compose it and non-zero), thereby negating potential impact.

However, as IndexedMapping.sol checks for existence of a stored record before deleting it (and subtracting from the length array of known records), concerns regarding this potential underflow are mitigated.

SOLIDIFIED

Audit Report for Bitgo November 19th, 2018.

Closing Summary

Bitgo and Kyber’s contracts contain only one minor issue, along with several areas of note. The contracts have been audited once before, and have thorough test coverage.

We recommend the minor issue is amended, while the notes are up to Bitgo's and Kyber's discretion, as they mainly refer to improving the operation of the smart contracts.

The DAO mentioned in WBTC's specifications was not in-scope for this audit, for testing purposes the assumption that all transactions come from a trusted working DAO was made.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Bitgo platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.