SOLIDIFIED

Audit Report for C-Layer on October 14th, 2019.

Summary

Audit Report prepared by Solidified for C-Layer covering the C-Layer token and C-Layer Oracle smart contracts (and their associated components).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on October 10th, 2019, and the final results are presented here.

Audited Files

The following contracts were covered during the audit:

https://github.com/c-layer/contracts/tree/audit-phase-1

Notes

The audit was based on commit `f344c2e0986951f1ce45a5757a54c425a1e69e09`, Solidity compiler version `0.5.12`.

Intended Behavior

C-Layer is a smart contracts platform that supports the tokenization of the following assets: Bonds, Equity, Payment and Utility.

SOLIDIFIED

Audit Report for C-Layer on October 14th, 2019.

Executive Summary

Solidified found that the C-Layer contracts contain several major and minor issues, in addition to several areas of note. As of October 23rd 2019, all the aforementioned issues were resolved by C-Layer in commit `audit-phase-1-final`.

Issues found:

Critical	Major	Minor	Notes
0	3	7	7

Issues Found

Major Issues

1. ChangeTokensale.sol: Potential mismatch between what off-chain investors pay and the number of tokens they receive (resolved)

If the number of tokens an off-chain investor has purchased exceeds the current availableSupply() at the time the transaction is executed, addOffchainInvestment() will only distribute a fraction of the tokens purchased to them.

Recommendation
Have addOffchainInvestment() revert if availableSupply() is less than _amount.

Amended [October 23rd 2019]
`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

2. BonusTokensale.sol: earlyBonus() will fail after bonus period is over (resolved)

In L60 in BonusTokensale.sol, the calculated id variable will only be less than bonuses_.length until the currentTime() is less than bonusUntil_. Once the bonus period is over, this function call will result in an “array out of bounds” error, thus it will not be possible to buy any tokens.

Recommendation
Return zero in earlyBonus() as soon as currentTime() is greater than bonusUntil_.

Amended [October 23rd 2019]

`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

3. BonusTokensale.sol: firstBonus() will fail if totalRaised_ is greater than bonusUntil_ (resolved)

In L72 in BonusTokensale.sol, the calculated id variable will only be less than bonuses_.length until the totalRaised_ < bonusUntil_ (note, that the comparison is being made between Wei and Epoch ticks).
Once the totalRaised_ amount in Wei gets bigger than bonusUntil_ Epoch ticks, the function call will result in an “array out of bounds” error, thus it will not be possible to buy any tokens.

Recommendation
Fix the first bonus formula to not exceed the bonuses array bounds.

Amended [October 23rd 2019]
`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

Minor Issues

4. BaseTokensale.sol: Investors can potentially lose funds if withdrawAllETHFunds() is called prematurely (resolved)

Investors with unspentETH can potentially lose all their unspent ETH funds if withdrawAllETHFunds() is called before they had the chance to withdraw their funds.

Furthermore, calling withdrawAllETHFunds() does not currently reset totalUnspentETH_, which can cause investInternal to break.

Recommendation

Consider either:

A. Having this function only withdraw address(this).balance - totalUnspentETH_.
B. Creating an explicit deadline after which investors will lose all their non withdrawn ETH funds. withdrawAllETHFunds() should then revert if called before this deadline has passed. This deadline should be communicated to all respective parties.

Amended [October 23rd 2019]

`audit-phase-1-final`

C-Layer explained that this function is only to be called under critical conditions where the contracts are compromised due to unforeseen circumstances. Once called, C-Layer's intent is to have all the unspent ETH funds refunded back to their respective investors.

5. WithClaimsTokenDelegate.sol: Potential for denial of service (DoS) on transfer() and transferFrom() (resolved)

Both transfer() and transferFrom() call the hasClaims() function, which iterates over the tokens_[_token].claimables array. If enough claims are assigned to this array, the gas used by the function will eventually exceed the block gas limit. This will result in having the respective tokens in a "stuck" state where calling transfer() and transferFrom() always results in an error denoting that the block gas limit has been exceeded.

Recommendation

In defineClaimables(), place an upper limit on the number of claims that will never exceed the block gas limit.

Amended [October 23rd 2019]

`audit-phase-1-final`

C-Layer created documentation instructing operators to not add a number of claims that would cause the block gas limit to be exceeded.

6. OperableCore.sol: Functions assignOperators() and assignProxyOperators() missing _role validation (resolved)

The functions assignOperators() and assignProxyOperators() do not check if _role has been defined in roles before assigning it to the given operators.

Recommendation
Document this if it is the intended behavior, otherwise add the appropriate validation logic.

Amended [October 23rd 2019]
`audit-phase-1-final`
C-Layer documented that this was the intended behavior.

7. BaseTokenDelegate.sol: Function canTransfer() does not check for invalid _from addresses (resolved)

canTransfer() returns TransferCode.OK even when _from is invalid (e.g. address(0)).

Recommendation
Add appropriate validation logic.

Amended [October 23rd 2019]
`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

8. RatesProvider.sol: Incorrect array length validation in defineRatesExternal() (resolved)

rates.length is incorrectly validated to be <= currencies.length. Since the _rates array does not include the base currency, its length should always be shorter than currencies_. Furthermore, a more relevant validation would be to ensure the length of _rates does not exceed the length of rates_.

Recommendation
Change validation statement to:
`require(_rates.length <= rates_.length, "RP03");`

Amended [October 23rd 2019]
`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

9. TokenERC20.sol: ERC-20 incompatibility (resolved)

The constructor assigns totalSupply to the balance of the _initialAccount. This action should emit a Transfer (from = 0x0, to = _initialAccount, amount) event.

Amended [October 23rd 2019]
`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

10. Functions declare a return value but return none (resolved)

The following functions declare a return value but actually return none:

- FreezableTokenDelegate.sol: Function freezeManyAddresses().
- TokenCore.sol: Function defineOracles().
- BonusTokenSale.sol: Function defineBonus().
- BaseTokensale.sol: Function withdrawAllETHFunds().
- UserRegistry.sol: Function suspendUser().

Amended [October 23rd 2019]
`audit-phase-1-final`
The issue was fixed and is no longer present in the contract.

Notes

11. Eliminate code duplication between c-layer-core and c-layer-oracle

Identical files such as SafeMath.sol, Ownable.sol and Operable.sol should ideally not be duplicated across the different code bases.

12. UserTokensale.sol: Hard-coding contributionLimits limits reusability of UserTokensale (resolved)

Consider providing the contributionLimits values at initialization time.

13. UserRegistry.sol: Avoid repetition of validation code

The validation code require(_userId > 0 && _userId <= userCount_, "UR01"); has been repeated numerous times within UserRegistry. It's best practice to isolate this code into its own modifier or function.

14. RatesProvider.sol: rateOffset_ needs better documentation (resolved)

Please provide better documentation for rateOffset_ and whether its value is reflected when providing values in defineRates().

15. LimitableReceptionTokenDelegate.sol: audit variable declared as memory instead of storage (resolved)

Since audit is referencing data residing in storage, it is more efficient to declare it as a storage variable.

16. SchedulableTokensale.sol: misleading parameter name in investInternal() (resolved)

Consider renaming _refundUnspentETH to _exactAmountOnly.

17. Function auditUser() return variable names order mismatch in between TokenCore.sol and ITokenCore.sol (resolved)

Fix declaration order for lastEmissionAt, lastReceptionAt, cumulatedReception and cumulatedEmission.

SOLIDIFIED

Disclaimer