SOMISH

Blockchain Labs

Smart Contract Audit Report

Zillios Crowdsale

September 4th, 2019

Index

Summary

Audit Report prepared by Somish Blockchain Labs for Zillios crowdsale and token smart contracts.

Process and Delivery

Two (2) independent experts performed an unbiased and isolated audit of the code below. The debrief took place on September 4th, 2019, and the final results are presented here.

Audited Files

The following contract(s) were covered during the audit:

- ZilliosCSandToken.sol.

Client Documents

The following document(s) were provided by the client for the purpose of the audit:

- Zillios_Lightpaper dated June 2019.
- Zillios_Whitepaper dated June2 2019.

Notes

Audit was performed on email attachment received from Zillios and pragma version 0.5.7

Intended Behavior

The launch of Zillios and the corresponding Zillios token creation process shall be organized around smart contracts running on the Ethereum platform. The minimum amount of the token distribution event shall be set at $2.5 million; the maximum amount shall be set at $10 million. Zillios will be accepting Ethereum. The pre-sale and main sale event will run for a restricted period of time, or until the hard cap is reached. If the soft cap will not be reached, funds will be returned.

The symbol of the Zillios token is called " ZLST ", there will be a limited amount of tokens created and no more tokens will be issued after the sale. The total token supply is fixed at 1.000.000.000 (1 billion) ZLST tokens.

40% of the total 1 billion ZLST tokens will be allocated to the pre-sale and main-sale contributors who participated in the token sale, including bonuses provided to participants. Discount price and vesting period will be shared and announced when the pre-sale starts.

25% of the ZLST tokens will be allocated for the company's reserve and will have a total vesting period of 2 years with a 12 month cliff. During the cliff, every month 1/12 part of the reserve will be unlocked. The reserve will be used to grow the ecosystem and future development within the Zillios platform.

25% of the ZLST tokens will be allocated to the founding team of Zillios and employees. The founding partners and the team have been working on Zillios since 2017 and will have a vesting period of 2 years after the end of the token sale with a 6-month cliff. These tokens will be locked in the smart contract and will be released 1/6 part per month during the 6 month cliff period to avoid large fluctuations on the Zillios ecosystem.

3% of the ZLST tokens will be allocated to advisors. The advisors of Zillios will have a total vesting period of 12 months, with a 6-month cliff.

6% of the ZLST tokens will be allocated to rewards for the Zillios ecosystem. To incentivize early adopters a limited amount of tokens will be available for stakeholders to reward specific actions that drive platform growth.

1% of the ZLST tokens will be allocated to bounties with a 4 week vesting.

Issues Found

Major

1. Initial supply is unallocated

As per ERC-20 standard, the variable `totalsupply` represents the total number of tokens in supply, meaning sum of balances of all addresses. In case of a fixed supply token, `totalsupply` of token should be equal to the balance of owner, i.e. in this case Crowdsale contract which should be equal to initial supply, in this case 1 billion tokens. It is a standard practice to initially allocate the initial supply to token owner's account and then transfer tokens to the purchasers/founders/advisers etc. Once the token sale event is complete, all extra tokens in the owner's account should be burned.

2. Token allocations with lock should be done at the beginning of sale

Authority to grant tokens to advisor, teamFounder and company reserves is given to the owner, that too post the lock period ends. If the owner forgets or willingly doesn't allocate these tokens, they will become unused and then burnt. Ideally, these tokens should be allocated to respective holders at the beginning of sale with an individual lock for their respective vesting schedules. It is then, at the discretion of the holder to withdraw his tokens post the lock period.

3. Cliff concept is not implemented

The light paper mentions the concept of token vesting along with cliffs. For example, tokens for founders shall be vested for 48 months with a cliff of 12 months. However, we could not find an implementation of lock + cliff concept in the smart contract provided.

Recommendation
Reuse TokenVesting.sol from OpenZepplin's open-source contracts available at https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/drafts/TokenVesting.sol

4. Total supply is not updated when tokens are minted

Minting of tokens means new tokens being added to the supply. Since this is a case of fixed supply, tokens need to be minted at the time of contract creation only. Hence, the function `mint` should only be called in the `constructor`. Post that, tokens should be

transferred to purchasers instead of calling the mint function. Additionally, when tokens are minted the transfer event should be initiated from zero address to receiver.

5. Tokens should be burnt from specific account

Similar to point 4, burning of tokens means that tokens are subtracted from someone's account and the total supply. In this case, the unsold tokens when burnt, should reduce tokens from the owner's, (i.e. CrowdSale contract) address as well as the total supply. Additionally, the function should call Transfer event from owner address to zero address.

6. Missing check for WEI_500_USD in getTokenAmount

WEI_500_USD variable is updated through setWEIPrice function, which is not called while the contract is created. This could result in initiation of the crowdsale event with everyone getting 20% bonus.

Recommendation
Add a check for WEI_500_USD>0 in function buyTokens.

7. Time lock for allocating tokens is increased for every grant

Time lock is increased by 30 days while granting tokens to advisor, team founder and company reserves, which means founder 2 can be allocated tokens only 30 days after founder 1 has been allocated tokens. This seems to be an incorrect implementation.

8. ZIlliosToken totalSupply shadows ERC20Basic totalSupply

totalSupply is re-declared in ZIlliosToken contract which is a duplicate for existing totalSupply in parent contract i.e ERC20Basic. Because of this, functions like burnToken are not working.
Recommendation
Assign totalySupply without declaring in ZilliosToken contract.

Minor

9. Undeclared variable

icoSupply variable is undeclared in burnToken() function of Allocations contract. This leads to compilation errors.

Recommendation
Use variable publicSupply instead

10. Usage of deprecated denomination

The denomination years is used for setting timeLock variable, which is deprecated after solidity version 0.5.0.
Line numbers: 232, 233, 234

Recommendation
Convert unit to days

11. Crowdsale start time and end time check

Crowdsale ending time must always be greater than starting time but **greater than or equal to** check is placed in Crowdsale constructor, which could possibly end up crowdsale without even starting.

12. Ownable.sol - OwnershipTransferred event not raised in constructor

Event OwnershipTransferred should be raised in constructor as well. Such that initial owner address is logged through event.

13. Safemath library

a. Inappropriate assert condition a==0 in mul function, it should return zero if any of the input is zero.
b. No check added for zero denominator in div function, which may revert back the transaction.

Recommendation : Re-use SafeMath library from OpenZeppelin github repository at https://github.com/OpenZeppelin/openzeppelin-contracts

14. Missing check for zero addresses

Any tokens issued to zero address are permanently locked and result in erroneous token supply. A check for zero addresses (0x00...000) should be added to the following functions:

a. Standard Token - function approve()
b. MintableToken - function mint()

15. setWEIPrice() and changeRate() can be called by owner anytime

Ideally token rate should not be changeable after crowdsale has started, there is no such check that restricts owner to change rate after token sale has started.

Recommendation
In case you need to change the rate as per the latest ETH->USD rate, you can use the DAI feed instead of manually posting rates.

16. onlyOwner check added for finalizeCrowdsale()

Since time check has already been added in finalizeCrowdsale() function, the contract should not rely only on the owner to call the function.

17. totalsupply is duplicated in crowdsale

The variable Total supply is already declared in token contract, so there is no need to store in crowdsale contract. It causes unnecessary confusion and doubles the operation costs. Eg. When tokens are burned, both the supplies are reduced.

18. No individual cap is applied

The whitepaper talks about an individual cap per contributor. However, no implementation of minimum and maximum amounts per contributor was found in the smart contract.

Notes

1. **Write Unit test cases**

   The contract provided had compilation errors along with errors like function burnToken not working. Such errors could have easily been avoided by writing automated test cases. In general, we would recommend writing unit-test cases to achieve 100% line and branch coverage in order to avoid any unforeseen issues.

2. **Re-use open source libraries**

   Most of the requirements could have been met re-using audited open source smart contracts like OpenZepplin available at

https://github.com/OpenZeppelin/openzeppelin-contracts. We recommend re-using these rather than building from scratch.

3. Usage of Safemath library

Consider following same pattern while using safemath library. In some of arithmetic operations SafeMath function is explicitly called and some operations are carried normally.

Recommendation

Since `using SafeMath for uint256` is used, no need to call `SafeMath` function explicitly.

4. Implementation of Crowdsale interface

No need of inheriting Crowdsale contract in `ZillosCrowdsale` contract as it is already inherited in `CappedCrowdsale` and `RefundableCrowdsale` contracts

5. Code commenting

Consider adding comments to functions and variable declarations in the smart contracts as it gives clarity to the reader and reduces confusion for the developers as well. Comments are missing in all contracts.

6. Hard-coded constants and types

Consider using constants for constant values instead of hardcoding. For example **teamFounderSupply**, **companyVestingSupply** in **Crowdsale** contract.

Recommendation

For readability purposes, the usage of a constant type is suggested.

7. Add reasons for revert

Consider adding messages for all **require** or **revert** statements; it gives clarity to the users and helps easy debugging.

8. Follow the Solidity style guide

Consider following the Solidity style guide. It is intended to provide coding conventions for writing solidity code.

Recommendation

You can find documentation for Solidity Style guide on the following link: https://solidity.readthedocs.io/en/v0.5.7/style-guide.html. Use solhint tool to check for linting errors.

9. Gas Optimizations

Consider using keywords like **internal** and **external** instead of **public,** it will reduce gas consumption.

Recommendation
Consider using **internal** for the function which is callable only within the same contract and using **external** for the function which is callable only from outside of contract. It will optimize the function call as well as gas required for deployment.

Closing Summary

Upon audit of the Zillios's smart contract, it was observed that the contracts contain major and minor issues, along with several areas of notes.

We recommend that the major and minor issues should be resolved. Resolving for the areas of notes are up to Zillios's discretion. The notes section refers to improving the operations of the smart contract.

Disclaimer

Somish Blockchain Labs's audit is not a security warranty, investment advice, or an endorsement of the Zillios's platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. 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 Somish from legal and financial liability.

Somish Solutions Limited