SOMISH

Blockchain Labs

Smart Contract Audit Report

AsureNetwork Crowdsale

July 4th, 2019

Index

Summary

Audit Report prepared by Somish Blockchain Labs for AsureNetwork smart contract.

Process and Delivery

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

Audited Files

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

- AsureBonusesCrowdsale.sol.
- AsureBounty.sol.
- AsureCrowdsale.sol.
- AsureCrowdsaleDeployer.sol.
- AsureToken.sol.
- Migrations.sol.

Client Documents

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

- Main - Document Outlining Important details of Asure Network Token Generation Event dated 18th June 2019.

Notes

Audit was performed on commit **4098f931508d99fae1fc52849c40180d3329a3b7** and pragma version ^0.5.0

Intended Behavior

The ASR TGE is intended to happen in two rounds. The first round will take place in mid-2019. The second round will happen later in 2019. Tokens will be ERC20 compatible and limited in supply by 100.000.000. In total, 45% of all ASR utility tokens ("ASR") will be sold through the TGE.

The ASR token is a utility token. It is implemented as an Ethereum smart contract and supports the ERC-20 token standards. The token will be used by Network Validators as well as Service

Providers to participate as stakers in the proof-of-stake consensus mechanisms of the Asure network. It shall act as an incentive for all stakeholders to work correctly. Also, the ASR token will be used to govern the Asure network.
The table below highlights how the investments will be allocated.

- Phase 1: 20% of all ASR tokens will be generated.
- Phase 2: 80% of all ASR tokens will be generated.

Phase 1

In the first phase, 20% of all ASR token will be generated.

10%	Public Pre-Sale	Contributions will be used to develop the net-work minimal viable product, and to build big-ger community.
5%	Family & Friends	Family and Friends receive their tokens as part of their compensation package.
5%	Bounty	Asure provides compensation for a number of tasks spread across marketing, bug reporting or even improving aspects of the Asure network, blockchain and platform.

Phase 2

In the second phase, 80% of all ASR tokens will be distributed.

35%	Public Main-Sale	Contributions will be used to develop the plat-form, and to fund security, legal and operational needs.
35%	Foundation	Comprises foundation development and education initiatives, incentives to developers and to

		research blockchain, scaling, network, and plat- form.
8%	Team	These are placed to acknowledge the time, e ort and resources contributed to the Asure plat- form. The Asure team receives their tokens as part of their compensation package, and team tokens will be vested.
2%	Advisors	Advisors receive their tokens as part of their compensation package.

The Asure Team and Advisors will receive their tokens over two years after the start of the second phase. The vesting ensures token course stability and commitment of all involved members. If a holder attempts to transfer more ASR tokens than vested, the transaction will be blocked.

Whitelisting

Participation in the TGE is only possible for the whitelisted participants who have carried out the KYC process.

Issues Found

Critical

1. Tokens withdrawable by owner before the end of sale

The function transferToIEO() in AsureCrowdsale allows the owner to transfer any amount of tokens to any address while the sale is on. This can result in all excess tokens being transferred to another account instead of being burnt or excessively increasing the %age holdings of investors.

Recommendation

If tokens need to be transferred they should be done in the function mint() in **AsureCrowdsaleDeployer.sol**.

Major

1. Missing individual check for Team members and Advisors tokens.

While the contract **AsureCrowdsaleDeployer.sol** does maintain constants **AVAILABLE_ADVISOR_SUPPLY** and **AVAILABLE_TEAM_SUPPLY**, there are no individual checks ensuring that team members will receive token = **AVAILABLE_TEAM_SUPPLY** and advisors will receive **AVAILABLE_ADVISOR _SUPPLY** tokens.

It is ensured that Team members + Advisors = 10% of total supply, but there is a possibility that the individual proportion is not as per the amounts declared(Team members = 8 Advisors=2). Ex:- Team members 7%, Advisors 3%.

2. Missing check to ensure token vesting for Team Members and Advisors.

There is no check found to ensure that vesting has been enforced on team members and advisors. Assuming open zeppelin's **TokenVesting.sol** shall be deployed for vesting team members and advisors tokens, there should be a check in **mint()** function in **AsureCrowdsaleDeployer.sol** to check **duration()** parameter for each advisor/team member address specified.

Minor

1. Crowdfund investment are being transferred to an unknown address

The funds are being transferred to a wallet address. Since nothing has been mentioned about this address, it may be a concern for investors. This address may/may not be a smart contract. We would recommend using a Gnosis multi-sig wallet here.

2. ERC-20 transfers should be wrapped in require statements

The ERC20 standard defines that the **transfer()** function should return a boolean and although most implementations throw on failure, this behavior is not required. Therefore it

is recommended to wrap transfer calls in require. You can also replace it with **safeTransfer()** from **SafeERC20.sol** as it is already wrapped in require statement.

Line numbers: AsureBounty.sol 20,26
AsureToken.sol 45

3. Missing check in mint(),transferFrom() functions

There should be a check for self transfer/mint in **mint(), transferFrom()** functions. The same check which is available in **transfer()** should be added here**.** i.e, **require(to != address(this));**

Notes

1. 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 few contracts.

2. Hard-coded constants and types

Consider using constants for constant values instead of hardcoding. For example line 26 in **AsureBounty** contract, line 63 in **AsureCrowdsale** contract.

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

3. Add reasons for revert

Consider adding messages for **require** or **revert** statements, it gives clarity to the users.
Line numbers: AsureBonusesCrowdsale.sol 74 - 77
AsureBounty.sol 17
AsureCrowdsale.sol 42,53,63
AsureCrowdsaleDeployer.sol 40,41,53,66,92
AsureToken.sol 29

4. 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/v0.4.24/style-guide.html. Use solhint tool to check for linting errors.

5. 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.

6. Replace if - revert patterns with require

Consider using **require()** statement instead of **revert()** inside **if** condition.
Line number: AsureToken.sol 42

Closing Summary

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

We recommend that the critical, major and minor issues should be resolved. Resolving for the areas of notes are up to AsureNetwork's discretion. The notes refer 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 AsureNetwork'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