Atonomi Contract Audit Report

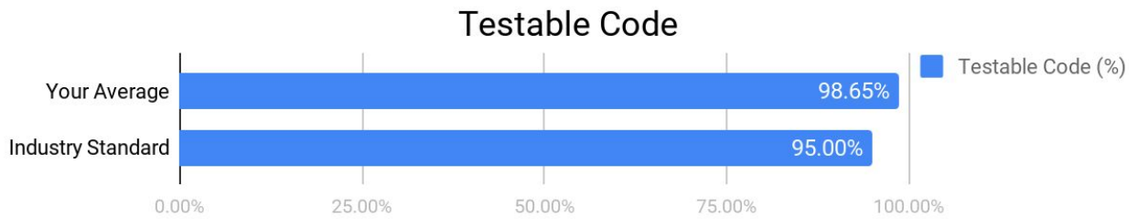
Prepared by Hosho
July 6th, 2018

Report Version: 2.0

Executive Summary

This document outlines the overall security of Atonomi's smart contract as evaluated by Hosho's Smart Contract auditing team. The scope of this audit was to analyze and document Atonomi's contract codebase for quality, security, and correctness.

There is a single low severity issue that exists within these contracts. (See Complete Analysis)



Testable code is 98.65% which is greater than the industry standard of 95%. (See Coverage Report)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Hosho recommend that the Atonomi team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table Of Contents

1. Auditing Strategy and Techniques Applied 3

2. Structure Analysis and Test Results 4

2.1. Summary

2.2 Coverage Report

2.3 Failing Tests

3. Complete Analysis 5

3.1 Unresolved, Low: Incorrect Definition

4. Closing Statement 6

5. Appendix A 7

Test Suite Results

6. Appendix B 16

All Contract Files Tested

7. Appendix C 17

Individual File Coverage Report

1. Auditing Strategy and Techniques Applied

The Hosho team has performed a thorough review of the smart contract code, the latest version as written and updated on June 29th, 2018. All main contract files were reviewed using the following tools and processes. (See All Files Covered)

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing ERC-20 Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks; and
- Is not affected by the latest vulnerabilities.

The Hosho team has followed best practices and industry-standard techniques to verify the implementation of Atonomi's contract. To do so, the code is reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1. Due diligence in assessing the overall code quality of the codebase.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.
5. Deploying the smart contract to testnet and production networks using multiple client implementations to run live tests.

2. Structure Analysis and Test Results

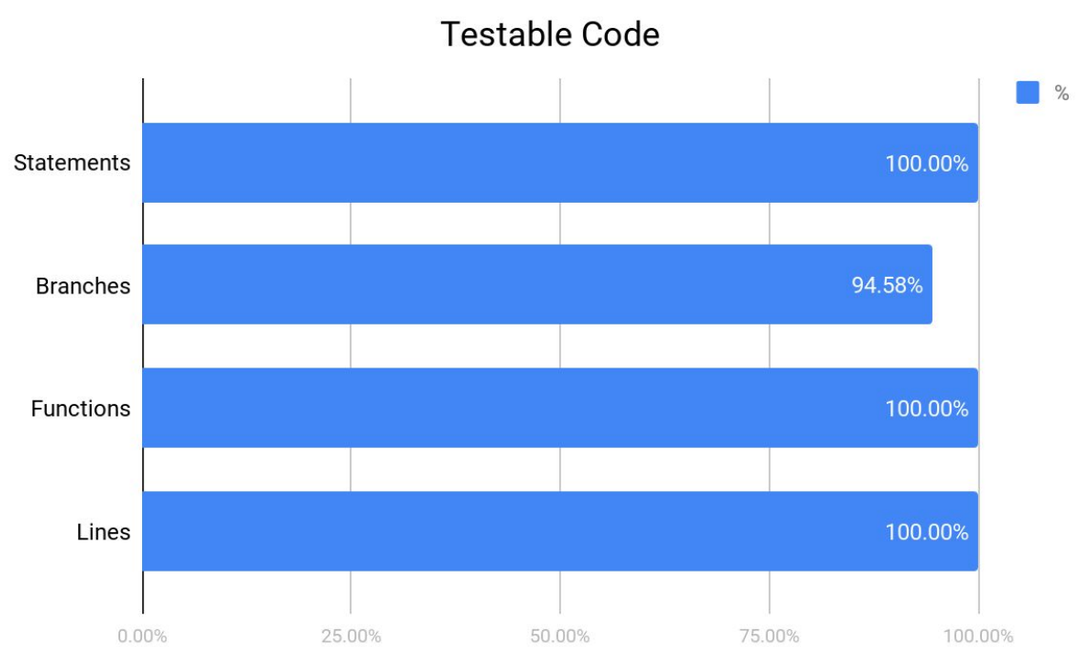
2.1. Summary

Atonomi is a ledger and reputation system for tracking manufacturers and the individual devices that they produce. The Atonomi team has also incorporated a reputation system that allows decentralized ratings for manufacturers as well as an ERC-20 token used to deliver rewards to the manufacturers, based on their reputation.

The only issue that remains is a low risk issue that prevents perfect ERC-20 compliance but will not prevent normal contract operations.

2.2 Coverage Report

As part of our work assisting Atonomi in verifying the correctness of their contract code, our team was responsible for writing a unit test suite using the Truffle testing framework.



For each file see Individual File Coverage Report

2.3 Failing Tests

1. Ensure 'AMLToken' defines the ERC-20 Token Standard Interface. Should have the correct 'decimals' definition. (See Issue 3.1)

See Test Suite Results for all tests.

3. Complete Analysis

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged "Resolved" or "Unresolved" depending on whether they have been fixed or still need addressing. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

- **Critical** - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.
- **High** - The issue affects the ability of the contract to compile or operate in a significant way.
- **Medium** - The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.
- **Low** - The issue has minimal impact on the contract's ability to operate.
- **Informational** - The issue has no impact on the contract's ability to operate, and is meant only as additional information.

3.1 Unresolved, Low: Incorrect Definition

Contract: CrowdsaleToken

Explanation

The ERC-20 standards state that the declaration for `decimals` should be a *uint8* as opposed to *uint256* as it currently is in this contract.

---

4. Closing Statement

The Hosho team is grateful to have been given the opportunity to work with the Atonomi team.

The team of experts at Hosho, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, can say with confidence that the Atonomi contract is free of any critical issues, having passed the rigorous Hosho auditing process.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

We at Hosho recommend that the Atonomi team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

5. Appendix A

Test Suite Results

Contract: CLIENT CONTRACT

√ Should reclaim unreleased tokens (101ms)

√ Should fail after token has been released (237ms)

√ Should allow transfer of contract to new owner (67ms)

√ Should now allow `transfer` to account 0 (59ms)

√ Should fail if not called by owner (62ms)

Contract: ERC-20 Tests for AMLToken

√ Should deploy a token with the proper configuration (98ms)

√ Should allocate tokens per the minting function, and validate balances (527ms)

√ Should `transfer` tokens from *0xd86543882b609b1791d39e77f0efc748dfff7dff* to *0x42adbad92ed3e86db13e4f6380223f36df9980ef* (276ms)

√ Should not `transfer` negative token amounts (47ms)

√ Should not `transfer` more tokens than you have (48ms)

√ Should allow *0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3* to `authorize` *0x341106cb00828c87cd3ac0de55eda7255e04933f* to `transfer` 1000 tokens (78ms)

√ Should allow *0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3* to zero out the *0x341106cb00828c87cd3ac0de55eda7255e04933f* authorization (78ms)

√ Should allow *0x667632a620d245b062c0c83c9749c9bfadf84e3b* to `authorize` *0x53353ef6da4bbb18d242b53a17f7a976265878d5* for 1000 token spend, and *0x53353ef6da4bbb18d242b53a17f7a976265878d5* should be able to send these tokens to *0x341106cb00828c87cd3ac0de55eda7255e04933f* (214ms)

√ Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to `transfer` negative tokens from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* (44ms)

√ Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to `transfer` tokens from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* to *0x0*

√ Should not `transfer` tokens to *0x0* (41ms)

√ Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to `transfer` more tokens than authorized from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* (48ms)

√ Should allow an `approval` to be set, then increased, and decreased (258ms)

Ensure 'AMLToken' defines the ERC20 Token Standard Interface

√ Should have the correct `'name'` definition

√ Should have the correct `'approve'` definition

√ Should have the correct `'totalSupply'` definition

√ Should have the correct `'transferFrom'` definition

X Should have the correct `'decimals'` definition

√ Should have the correct `'balanceOf'` definition

√ Should have the correct `'symbol'` definition

√ Should have the correct `'transfer'` definition

√ Should have the correct `'allowance'` definition

√ Should have the correct `'Transfer'` definition

√ Should have the correct `'Approval'` definition

Contract: Tests for AMLToken

√ Should deploy a token with the proper configuration (68ms)

√ Should allocate tokens per the minting function, and validate balances (279ms)

Constructor tests

√ Should fail to construct the Atonomi contract with token at address 0 (545ms)

√ Should fail to construct the Atonomi contract with settings at address 0 (247ms)

√ Should fail to construct the `NetworkSettings` contract with 0 reg fee (103ms)

√ Should fail to construct the `NetworkSettings` contract with 0 actFee (85ms)

√ Should fail to construct the `NetworkSettings` contract with 0 rep reward (91ms)

√ Should construct the atonomi contract (465ms)

√ Should require that the `NetworkSettings` contract has a node share above zero, and below 101 (277ms)

Reputation Management

√ Should require a share above zero

√ Should require a share less than 101

√ Should require a share that differs from the current share (38ms)

√ Should allow the share to be set properly (69ms)

√ Should require that a new block threshold is different from the current

√ Should set a new block threshold (64ms)

Manufacturer Wallet Manipulation

√ Should require that the new wallet not be *0x0*

√ Should not allow the new account to be an IRN Admin (135ms)

√ Should not allow the new account to be a manufacturer (166ms)

√ Should not allow the new account to be an IRN Node (150ms)

√ Should not allow the new account to be registered with a member ID (137ms)

√ Should transfer all manufacturer states (124ms)

√ Should only set default reputation reward only if not previously set (1072ms)

√ Should not delete network member from pools if balance is set on member removal (1125ms)

Fee Setter tests

Activation Fee tests

√ Should set the activation fee to 1 (49ms)

√ Should fail set the activation fee to 0

√ Should fail set the activation fee to the same amount (88ms)

Registration Fee tests

√ Should set the registration fee (48ms)

√ Should fail to set the reg fee to 0

√ Should fail to set the reg fee to the same value (94ms)

Reputation Reward tests

√ Should set the reputation reward (52ms)

√ Should fail to set the reputation reward to 0 (39ms)

√ Should fail to set the reputation reward to the same value (85ms)

√ Should set the default reputation for a manufacturer (177ms)

√ Should fail set the default reputation for a manufacturer to the same value (218ms)

√ Should fail to set the default reputation for a manufacturer with ID *0x0* (38ms)

√ Should fail to set the default reputation for a manufacturer if called by a non-owner

√ Should calculate correct reward after reputation score update (1189ms)

Ownership Tests for AMLToken

√ Should not allow a non-owner to transfer ownership

√ Should not allow the owner to transfer ownership to *0x0*

√ Should allow the owner to transfer ownership

Device Onboarding tests

registerDevice() tests

√ Should register a device (236ms)

√ Should fail to register a device that's already been registered (280ms)

√ Should fail to register a device with hash 0 (52ms)

√ Should fail to register a device with type 0 (54ms)

√ Should fail to register a device without enough tokens (204ms)

√ Should fail to register a device from non-manufacturer

activateDevice() tests

√ Should activate a device (173ms)

√ Should fail to activate a device that has already been activated (228ms)

√ Should fail to activate an unregistered device (48ms)

√ Should fail to activate without enough tokens (149ms)

√ Should fail to activate if the own contract instance is the manufacturer (1419ms)

registerAndActivateDevice() tests

√ Should `registerAndActivate` a device (274ms)

√ Should fail to `registerAndActivate` a device without enough tokens (218ms)

Reputation Management Tests

updateReputationScore() tests

√ Should update the reputation score (231ms)

√ Should update the reputation score to the same value (305ms)

√ Should fail to update the reputation score of an empty ID (74ms)

√ Should fail to update the reputation score of an unactivated ID (304ms)

√ Should fail update the reputation score from non-IRNNode account

√ Should fail update the reputation score for unregistered `deviceID` (64ms)

√ Should fail to update reputation score if collector is the manufacturer (1641ms)

Bulk Operations tests

registerDevices() tests

√ Should register an array of devices (355ms)

√ Should fail to register an array of devices without enough tokens to pay fee (295ms)

√ Should fail to register a device with a hash of 0 (68ms)

√ Should fail to register an array of devices that has already been registered (453ms)

√ Should fail to register an empty array of devices (39ms)

√ Should fail to register an array of devices with a mismatching array of deviceTypes (202ms)

√ Should fail to register devices when hash array length does not match public key array length (50ms)

Whitelist Management tests

addNetworkMember Tests

√ Should `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as an IRNAdmin, Manufacturer, IRNNode with member id 1 (120ms)

√ Should `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as an IRNAdmin with member id 1 (121ms)

√ Should fail to `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as an IRNAdmin when it's already an IRNAdmin (134ms)

√ Should `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as a Manufacturer with member id 1 (121ms)

√ Should fail to `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as a Manufacturer when it's already a Manufacturer (144ms)

√ Should fail to `add` *0x3b44fa9f7511113a8c1a1528070d45b1d7cdd101* as a Manufacturer with ID *0x1* when *0x1* is already assigned to a manufacturer (186ms)

√ Should fail to `add` a manufacturer with ID 0 (59ms)

√ Should `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as an IRNNode with member id 1 (111ms)

√ Should fail to `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as a IRNNode when it's already a IRNNode (181ms)

√ Should fail to `add` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as a Manufacturer with id 2 it's already been given an ID (157ms)

removeNetworkMember tests

√ Should `remove` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as a network member (94ms)

√ Should fail to `remove` *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* as a network member when not called from an Owner or Admin

√ Should skip deleting manufacturer reward if network member is not a manufacturer (170ms)

√ Should skip deleting from pool if network member has no balance in the pool (208ms)

Token Rewards tests

√ Should handle token pool rewards (153ms)

√ Should handle token deposits (418ms)

withdrawTokens() tests

√ Should fail to `withdraw` 0 Tokens (51ms)

Dependency tests

Pausable tests

√ Should `pause` the contract (69ms)

√ Should `unpause` the contract (103ms)

TokenDestructible tests

√ Should destruct the contract and send the tokens back to the owner (231ms)

Wrapper coverage tests

validateAndRegister tests

√ Should fail to register a device that was already activated (305ms)

updateReputationScore tests

√ Should fail to update a reputation score on a device with a `mfgWallet` of 0 (306ms)

Contract: Tests for Atonomi's AML with all dependencies

AMLToken tests

√ Should transfer tokens to owner (85ms)

√ Should fail to transfer tokens to owner when called from a non-owner address (38ms)

BurnableToken tests

√ Should `burn` tokens (62ms)

MintableToken tests

√ Should set minting agent and mint new tokens (461ms)

√ Should not set minting agent when mint is finished (369ms)

√ Should fail to `mint` if not called by owner (367ms)

CrowdsaleToken tests

Crowdsale constructor Tests

√ Should construct a mintable Token (324ms)

√ Should construct an unmintable Token (323ms)

√ Should fail to construct an unmintable token with 0 supply (231ms)

ReleaseTokenTransfer() Tests

√ Should release token transfer (57ms)

√ Should fail to release token transfer from a non-release agent

canUpgrade() Tests

√ Should run `canUpgrade` and return true (142ms)

√ Should run `canUpgrade` and return false

setTokenInformation() Tests

√ Should set the token information (78ms)

√ Should fail to set token information from non-owner

[contract] tests

ReleaseTokenTransfer() Tests

√ Should release token transfer (97ms)

√ Should fail to release token transfer from a non-release agent

√ Should fail if token not released (47ms)

√ Should require the `owner` to equal the `release agent` (147ms)

canUpgrade() Tests

√ Should run `canUpgrade` and return true (121ms)

√ Should run `canUpgrade` and return false

setTokenInformation() Tests

√ Should set the token information (58ms)

√ Should fail to set token information from non-owner

Recoverable tests

√ Should recover tokens (833ms)

√ Should get return tokens (690ms)

StandardTokenExt

√ Should be token

Upgradeable Token

√ Verify `upgrade` agent interface

√ Should allow the upgrade master to be set (121ms)

√ Should allow the upgrade state to be checked (923ms)

√ Should require that the agent not be set to *0x0*

√ Should require that only the `upgradeMaster` can set the agent (42ms)

√ Should require a positive true return from `isUpgradeAgent` (88ms)

√ Should require that the `originalSupply` variable is the same as the current `totalSupply` (95ms)

Contract: SafeMath

√ Should skip operation on multiply by zero

√ Should revert on multiply overflow

√ Should allow regular multiple

√ Should revert on divide by zero

√ Should allow regular division

√ Should revert on subtraction overflow

√ Should allow regular subtraction

√ Should revert on addition overflow

√ Should allow regular addition

Contract: SafeMathLibTest

√ Should skip operation on multiply by zero

√ Should revert on multiply overflow

√ Should revert on addition overflow

√ Should allow regular multiple

√ Should revert on subtraction overflow

√ Should allow regular subtraction

√ Should allow regular addition

6. Appendix B

All Contract Files Tested

Commit Hash: bca6cf14ac6e77780c1dfe9b16a893ddbde3886a

File	Fingerprint (SHA256)
Atonomi.sol	C84070526102D871DF5FF39594ECE0340FF6AA2C7834FB50B3CE1931D0422AB2
NetworkSettings.sol	3210ADFC3C2AF6503D3BDAB879C08FB57B030E343ECA39FB33D96B7C3876EC21
tokenmarket/AMLToken.sol	3F34D9DF5F8979FC4109EB7FCE21FF724D6DD5942352BF09994ABD7D65BF0AD7
tokenmarket/SafeMathLib.sol	9569376492F82E6199B15E355D4687EE3D71F9C9B920F64FBA3A1165AA8D4269
zeppelin-solidity/contracts/lifecycle/Pausable.sol	78BF21E029FC3F1C38151915DB9CCCE2F0553BFEAE9B6685FDE1C297091CDB6F
zeppelin-solidity/contracts/lifecycle/TokenDestructible.sol	88BAF3D8ADACD686EB4C26735C7A286D18E42DF072151842D82ADE590B5F7FE7
zeppelin-solidity/contracts/ownership/Ownable.sol	394036F88497454E7A8B88FDEEAEBCA7023DCE7DAF81606C8A062FA117623BEB
zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol	66CFE33D9BEFA178964F04BC54CFCBF0F32131712BA2C8928E700E21EFB2C644

7. Appendix C

Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
Atonomi.sol	100.00%	91.27%	100.00%	100.00%
NetworkSettings.sol	100.00%	100.00%	100.00%	100.00%
tokenmarket/AMLToken.sol	100.00%	100.00%	100.00%	100.00%
tokenmarket/SafeMathLib.sol	100.00%	100.00%	100.00%	100.00%
zeppelin-solidity/contracts/lifecycle/Pausable.sol	100.00%	50.00%	100.00%	100.00%
zeppelin-solidity/contracts/lifecycle/TokenDestructible.sol	100.00%	100.00%	100.00%	100.00%
zeppelin-solidity/contracts/ownership/Ownable.sol	100.00%	100.00%	100.00%	100.00%
zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol	100.00%	100.00%	100.00%	100.00%
All files	**100.00%**	**94.58%**	**100.00%**	**100.00%**