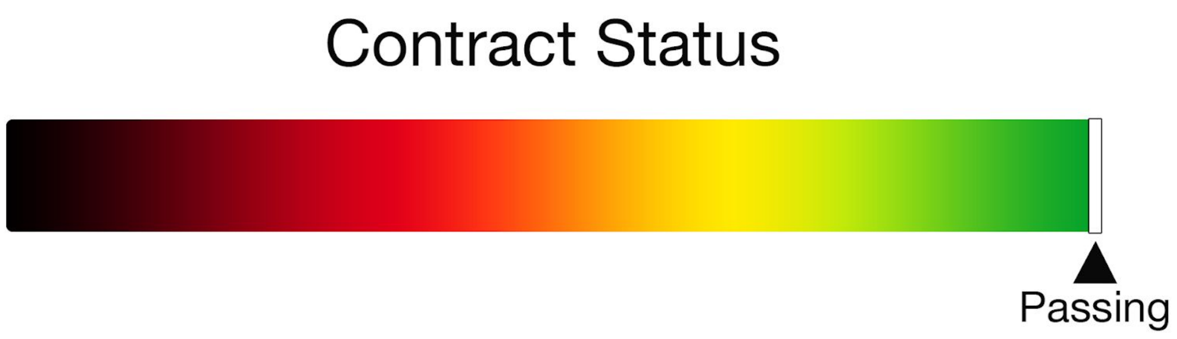
Wemark Contract Audit

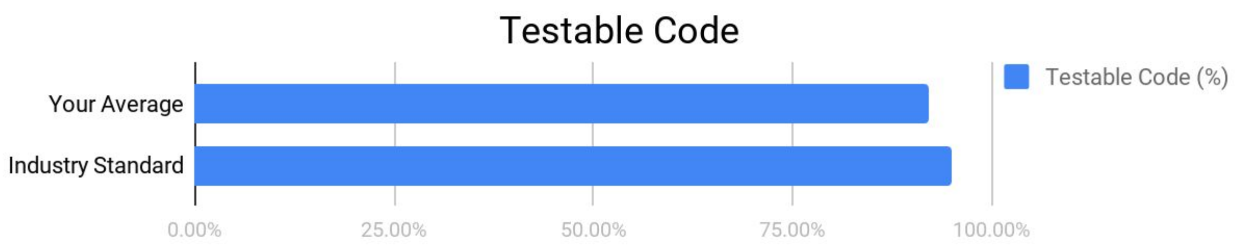
by Hosho, April 2018

Executive Summary

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



All issues have been remediated or acknowledged by the Wemark Team. (See Complete Analysis)



Testable code is on par with industry standard. (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 Wemark 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

The Hosho Team has performed a thorough review of the smart contract code, the latest version as written and updated on April 12, 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 token 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; and
- Uses methods safe from reentrance attacks.
- Is not affected by the latest vulnerabilities

The Hosho Team has followed best practices and industry-standard techniques to verify the implementation of Wemark's token contract. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they were discovered. Part of this work included 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.

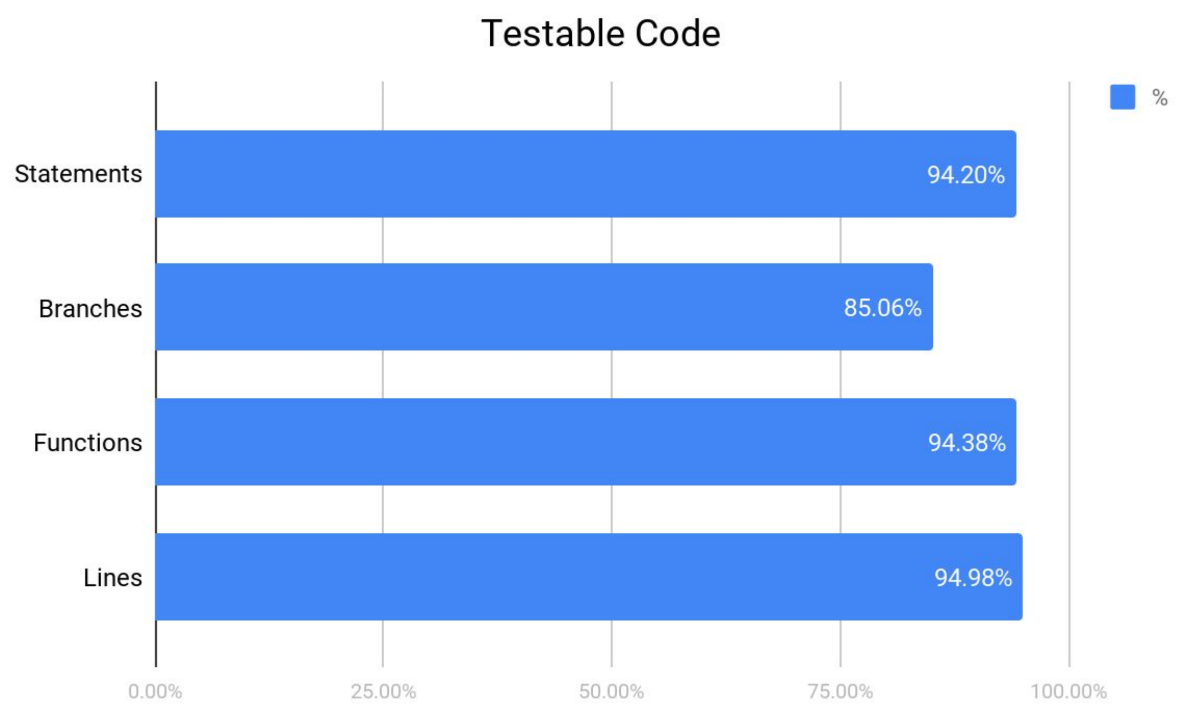
2. Structure Analysis and Test Results

2.1. Summary

The Wemark contracts consist of an upgradable, burnable, ERC-20 token, with a well implemented vesting system that is loaded from the Crowdsale. The Crowdsale contains a simple flat pricing structure, with bonus tokens provided based on the amount of ETH submitted to the contract.

2.2 Coverage Report

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



For individual files see Additional Coverage Report

2.3 Failing Tests

No failing tests.

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 addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

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

3.1. Resolved, Low: ERC-20 Compliance

WemarkToken

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.

Resolution

Acknowledged by the Wemark Team.

3.2. Resolved, Low: Unnecessary Inheritance

WemarkToken

Explanation

WemarkToken extends FractionalERC20, but the variable `decimals` is already declared in both WemarkToken and CrowdsaleToken. Additionally, the ERC-20 interface is enforced in CrowdsaleToken so nothing is gained by inheriting from FractionalERC20.

Resolution

The Wemark Team has removed the FractionalERC20 contract, eliminating the unnecessary inheritance.

3.3. Resolved, Low: Unusable Function

CrowdsaleToken

Explanation

During the initialization process, the function `setTokenInformation` is called to establish the token name and symbol. The WemarkToken child contract then explicitly sets both of these strings, overwriting anything established previously and rendering this function unusable.

Resolution

This has been acknowledged by the Wemark Team.

3.4. Resolved, Low: Invalid Check

WemarkCrowdsalePricingStrategy

Explanation

There is a check in this contract, at the beginning of the `calculatePrice` function, verifying that `milestoneCount` is greater than zero, in other words there is at least one milestone. However, as the initializer utilizes a `uint[10]` array then halves that length, `milestoneCount` cannot be smaller than five, making this check invalid.

Resolution

The invalid check has been removed from the `calculatePrice` function by the Wemark Team.

3.5. Resolved, Informational: Unnecessary Code

CrowdsaleToken

Explanation

The variables `name`, `symbol`, and `decimals` are overwritten through inheritance and thus are unused.

Resolution

As these variables are no longer being defined within the WemarkToken contract, passed in as parameters instead, there is no longer an issue with inheritance.

3.6. Resolved, Informational: Wasted Gas

WemarkTokenCrowdsale

Explanation

Altering `wemarkToken` to be a global variable would avoid gas costs from repeatedly setting it in the function.

Resolution

The `wemarkToken` variable has been declared as global which will avoid unnecessary gas costs.

4. Closing Statement

We are grateful to have been given the opportunity to work with the Wemark Team.

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

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 Wemark Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

5. Test Suite Results

Coverage Report: Contract: WemarkCrowdsale

√ Should deploy crowdsale with the proper configuration (1958ms)

√ Should handle whitelisting (1049ms)

√ Should only accept eth from whitelisted addresses (308ms)

√ Should handle preallocateVested (2477ms)

√ Should not finalize crowdsale (147ms)

√ Should preallocate tokens for crowdsale (829ms)

√ Should verify base contract (1545ms)

√ Should only accept eth from whitelisted addresses during presale (19793ms)

√ Should require that the finalize agent is deployed properly (1196ms)

√ Should handle time shifting for the end time (849ms)

Contract: Pricing Strategy

√ Should return the milestone data (189ms)

√ Should return back the token count based on milestones (1523ms)

Contract: WemarkToken

√ Should deploy a token with the proper configuration (298ms)

√ Should not transfer non-released token (379ms)

√ Should allocate tokens per the minting function, and validate balances (1150ms)

√ Should release tokens and enable transfer (832ms)

√ Should transfer tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (450ms)

√ Should not transfer negative token amounts (280ms)

√ Should not transfer more tokens than you have (307ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (147ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (147ms)

√ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (743ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (245ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b to 0x0 (89ms)

√ Should not transfer tokens to 0x0 (103ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer more tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (303ms)

√ Should allow an approval to be set, then increased, and decreased (495ms)

√ Should block addresses from transferring tokens (340ms)

√ Should not allow ETH to be sent to the contract (61ms)

√ Should return the upgrade state (80ms)

√ Should allow token information to be set (237ms)

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

1) 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: Additional token management

Vesting

√ Should require proper configuration for a vesting setup (11719ms)

√ Should allow revocation of a revoking enabled grant (3428ms)

√ Should allow calculation of the amount of tokens released during the vesting period (1258ms)

√ Should return the date when the last token can be transferred (1775ms)

Upgradeable Token

√ Should allow the upgrade master to be set (295ms)

√ Should allow the upgrade state to be checked (3219ms)

√ Should require that the agent not be set to 0x0 (193ms)

√ Should require that only the `upgradeMaster` can set the agent (159ms)

√ Should require a positive true return from `isUpgradeAgent` (450ms)

√ Should require that the `originalSupply` variable is the same as the current `totalSupply` (397ms)

6. All Contract Files Tested

Commit Hash: 38264317db19ab6c10f4399d7f1f871fbf847dfa

File	Fingerprint (SHA256)
contracts/WemarkCrowdsalePricingStrategy.sol	`80d1a8fe037422e8c4ab36562fcccddf792d0cdc9665afbdb4c51655bbc22dc5`
contracts/WemarkToken.sol	`84a68d803da54ecd22c418af8f0defe2657d9b22000513aa1a6790ea32885b62`
contracts/WemarkTokenCrowdsale.sol	`3a82072c7251fd214cf551eb7816e9daee96a19d5efc19a65fbe0cc0b9c3d8ea`
contracts/WemarkTokenCrowdsaleFinalizeAgent.sol	`eda0bc67b887723def04e183319daf9052e390a5f671003af3d2c2222ad43206`
contracts/lib/Haltable.sol	`cb527f970cf61bae9c82e5e96166ff7b4d616e5d2a15bb1eca6464e984a3f10f`
contracts/lib/crowdsale/Crowdsale.sol	`a4d5d489d2ec6eac68f8d3c0a4176a8485ae13822cb8cd69ba31024c02835e2a`
contracts/lib/crowdsale/CrowdsaleBase.sol	`5c98752269b1aeb1d63d8b0ac9c219bbe27f0f555547db451f4859e197946bbc`
contracts/lib/crowdsale/FinalizeAgent.sol	`5a6c2928e222dd8d8a1a2891330a837d55f50b8ef251582ad712c6b600a8c30d`
contracts/lib/crowdsale/PricingStrategy.sol	`336353613f823bdff38731f54cd550c73239c4a96ffd9d12fecbaec27f950554`
contracts/lib/token/BurnableToken.sol	`c251989596abbb86dbd9898dd02ba9846f14a08b47b5e32590208841c53eaad3`
contracts/lib/token/CrowdsaleToken.sol	`263d1a9797d76edacea7279a201ccf3bbd28f182567fc8022ca5b0d8eaba7afd`
contracts/lib/token/LimitedTransferToken.sol	`3a9cd0885ee52ea255f67f04c63607c1885ea6921b278479a001acbf624f17ae`
contracts/lib/token/ReleasableToken.sol	`b208e0637ac04b62f3c6bb244787a9bf01e894daa72d5cbb007c661d462d13f9`
contracts/lib/token/UpgradeAgent.sol	`df57182fa70918013398b2ccc13438beb4b8dbcd09262e8740f129db9318b399`
contracts/lib/token/UpgradeableToken.sol	`d8c040d13c955a578bf0223df39137b06c67133907b03e4968da5761ded1740e`
contracts/lib/token/VestedToken.sol	`65440b0e7501bca6a3ec5153834fea4d9e78fbdfb3ecba5625d12639948b306f`

7. Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
contracts/WemarkCrowdsalePricingStrategy.sol	100.00%	75.00%	100.00%	100.00%
contracts/WemarkToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/WemarkTokenCrowdsale.sol	100.00%	85.00%	100.00%	100.00%
contracts/WemarkTokenCrowdsaleFinalizeAgent.sol	100.00%	80.77%	100.00%	100.00%
contracts/lib/Haltable.sol	75.00%	66.67%	80.00%	75.00%
contracts/lib/crowdsale/Crowdsale.sol	100.00%	100.00%	100.00%	100.00%
contracts/lib/crowdsale/CrowdsaleBase.sol	85.14%	77.59%	86.67%	87.10%
contracts/lib/crowdsale/FinalizeAgent.sol	100.00%	100.00%	100.00%	100.00%
contracts/lib/crowdsale/PricingStrategy.sol	50.00%	100.00%	50.00%	50.00%
contracts/lib/token/BurnableToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/lib/token/CrowdsaleToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/lib/token/LimitedTransferToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/lib/token/ReleasableToken.sol	80.00%	75.00%	100.00%	84.62%
contracts/lib/token/UpgradeAgent.sol	0.00%	100.00%	0.00%	0.00%
contracts/lib/token/UpgradeableToken.sol	100.00%	100.00%	100.00%	100.00%

contracts/lib/token/VestedToken.sol	100.00%	100.00%	100.00%	100.00%
All files	**94.22%**	**84.66%**	**94.38%**	**95.00%**