Art Token Audit

By KrowdMentor, January 2018

KrowdMentor

Listened · Advised · Funded

Contents

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

Testability is high. All uncovered code is able to be manually tested. See Coverage Report.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, merely an assessment of its 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 KrowdMentor recommend that the Art Token Team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Auditing Strategy and Techniques Applied

The Krowdmentor Team has performed a thorough review of the smart contract code as written and last updated on January 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 standard 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 Krowdmentor Team has followed best practices and industry-standard techniques to verify the implementation of Art Token'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-compariso**n 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.

Structure Analysis and Test Results

There are a number of issues within the contracts, along with a few suggestions noted by the Krowdmentor Team that would create improvement overall. While the contracts themselves are functional, the issues detailed below with the lack of compliance causes major concerns with the token and ICO system.

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

The crowdsale repository was forked at https://github.com/marinalexandru/allpublicart and tests were added to address some of the issues raised above.

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:

- **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.

Contract Status

The token contract is fully ERC-20 compliant and follows industry standards.

Issues

All found issues are explained below. Their spread is:

Severity	Number of issues
High	2
Medium	2
Low	1
Informational	3

Unresolved, High: total supply check

The total supply check provided in AllPublicArtCrowdsale.buyTokens() might not work as intended: the total supply is not incremented when buy tokens is called but only when token.mint() is called, i.e. when *sendTokensToPurchasers* is called, presumably at a later time. Therefore, users participating in the crowdsale will not increase the total supply even though the supply could end up higher than the cap (TOTAL_SUPPLY_CROWDSALE) when the actual minting is done.

The total supply is incremented only when the actual minting is done, i.e. in *mintTokenForPrivateInvestors*, *mintTokensFor* or *sendTokensToPurchasers* but these do not check for the token cap.

Tests provided:

- *'audit: the crowdsale should be available to the normal buyer even if TOTAL_SUPPLY_CROWDSALE has been reached before start of crowdsale'*
- *'audit: the buyers should not exceed TOTAL_SUPPLY_CROWDSALE'*

Unresolved, High: twoPercent wallet address not checked for transfer to 0x0

There is no protection in place in *forwardFunds* to check that the percent wallet address is not 0x0 which can lead to loss of funds. We suggest in this case that the funds might be automatically transferred to the main wallet, adding a require to throw transactions or setting the twoPercent wallet in the AllPublicArtCrowdsale constructor.

Tests provided:

- *'audit: should not burn 2% if twoPercent is not set'*

Unresolved, Medium: Type mismatch

There are instances where addresses are either passed as arguments or declared as uint256 instead of address. This can lead to errors when types are checked, i.e. an address is 20 bytes and using uint256 can overflow an address value.

- AllPublicArtCrowdsale constructor: _whitelistRegistry, _apaBonus
- APABonus constructor: _whitelistRegistry
- WhitelistRegistry constructor: _preferentialRate

Unresolved, Medium: Missing checks for 0x0

There are instances where methods that take address arguments or use addresses do not use the necessary protections for address 0x0. This can lead to loss of ether, loss of tokens or to uninitialized member contracts:

- AllPublicArtCrowdsale.AllPublicArtCrowdsale - member contracts(wRegistry, apaBonus) could end up uninitialized and cannot be initialized outside of the constructor
- AllPublicArtCrowdsale.mintTokenForPrivateInvestors - can lead to loss of tokens
- AllPublicArtCrowdsale.mintTokensFor - can lead to loss of tokens

- LockTokenAllocation.addLockTokenAllocation - can lead to loss of tokens
- LockTokenAllocation.LockTokenAllocation - member contract (apa) could end up uninitialized. The address is checked in unlock() but it would be better checked inside the constructor since otherwise we could end up with it initialized to 0x0 and no way of setting it outside of the constructor.
- APABonus.APABonus - member contract (wRegistry) could end up uninitialized and cannot be initialized outside of the constructor

Tests provided:

- *'audit: should not let 0x or 0 addresses to be inputed in the constructor'*
- *'audit: should not let mintTokensFor() function execute with 0x address parameter'*

Unresolved, Low: validPurchase override

The *validPurchase* override does not check that the amount is not zero if the first term, i.e. *super.validPurchase*(), evaluates to *false*.

Unresolved, Informational: code reuse.

AllPublicArtCrowdsale.sendTokensToPurchasers should reuse the *hasEnded* instead of *now > endTime* in the second require if this require condition was intended to check that the crowsale has ended.

Unresolved Informational: Unnecessary check

AllPublicArtToken.burn() has an unnecessary check for value to be positive which is already unsigned. Unnecessary checks can waste gas and can decrease readability.

Unresolved, Informational: Missing comments or comments mismatch

Comments are missing

- AllPublicArtCrowdsale constructor: missing @param _apaBonus

Mismatch between comments and code:

- APABonus.calculatePreSaleBonus: comments specify different bonus for values higher than 35 ETH
- AllPublicArtCrowdsale.setTwoPercent: @param beneficiaryAddress specifies wrong percentage
- AllPublicArtCrowdsale.mintTokenForPrivateInvestors(): comments suggest that the minting can only be done “before crowdsale starts” but there is no check provided to that effect

All Contract Files Tested

The audited files are:

1. APABonus.sol
2. AllPublicArtCrowdsale.sol
3. AllPublicArtToken.sol
4. LockTokenAllocation.sol
5. Migrations.sol
6. WhitelistRegistry.sol

Closing

We are grateful to have been given the opportunity to work with the Art Token Team. There are multiple suggestions from the KrowdMentor Team that will improve overall code quality and stability once implemented.

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 KrowdMentor recommend that the Art Token Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties