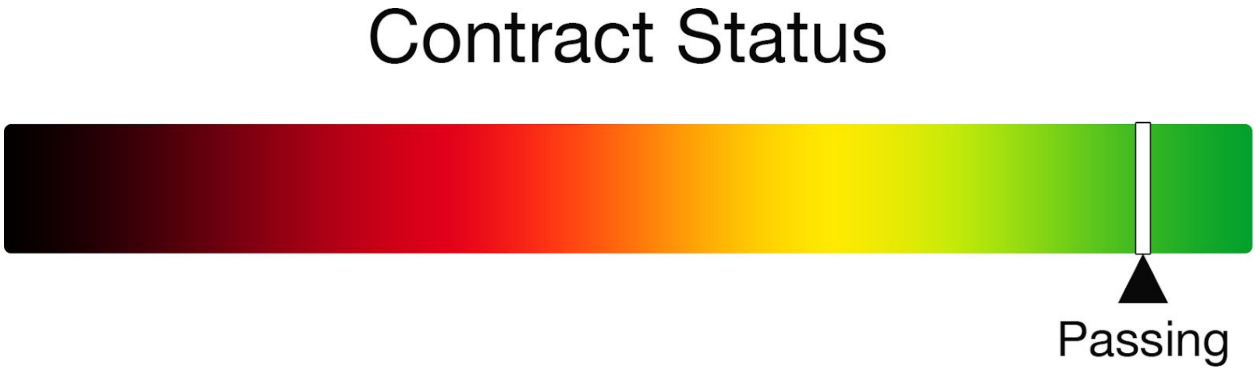
Decentralized Machine Learning Contract Audit

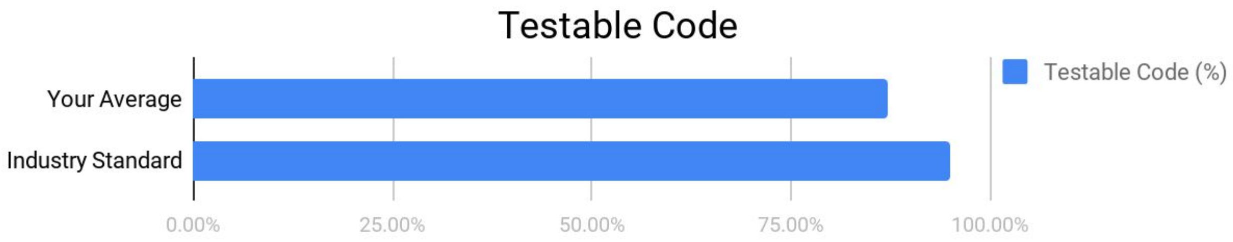
by Hosho, April 2018

Executive Summary

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



All issues have been remediated or acknowledged. (See Complete Analysis)



Testable code is lower than 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 DML 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 11, 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 DML'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

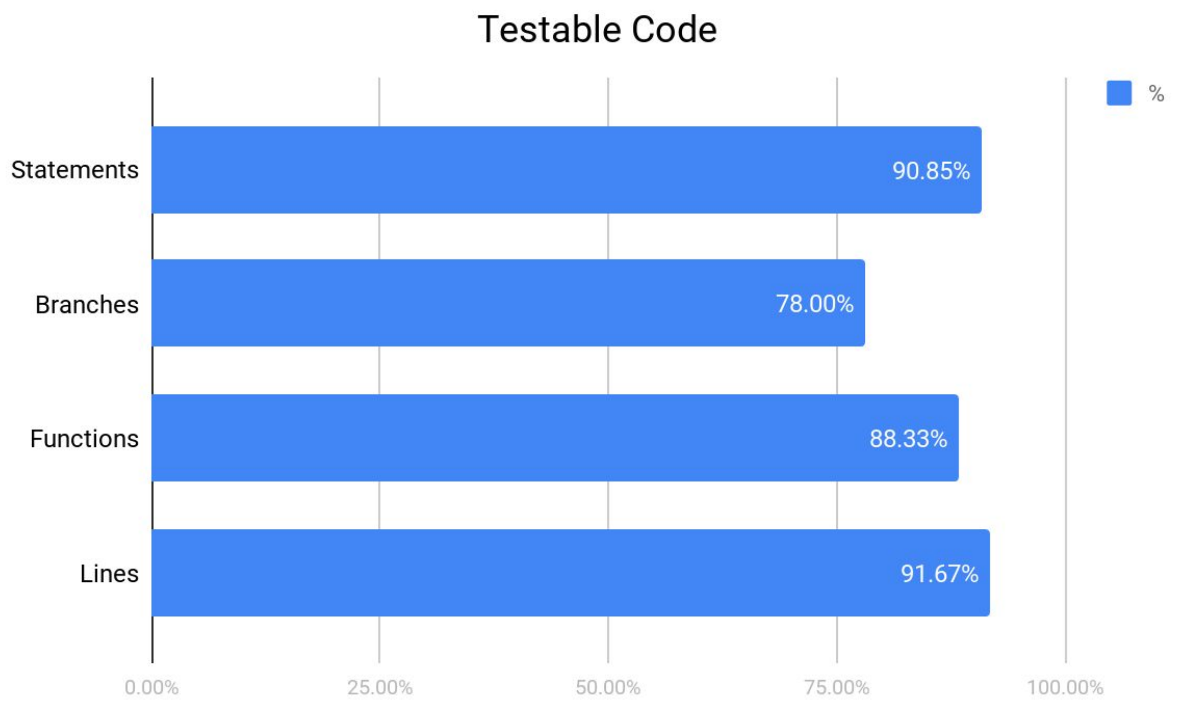
2.1. Summary

The DML contracts comprise a modified, ERC-20 compliant token and crowdsale that both contain extended functionality. The token includes minting capabilities with stages that are able to be unlocked by the owner, the crowdsale contract. The crowdsale contains a tranche based pricing system that allows for stable allocations both inside and outside of the whitelist system, normalized towards the main token price.

Due to the non-needed checks, as well as the non-needed modifiers, the overall coverage is low on these contracts. Additionally, since these contracts have non-hittable code paths, branch coverage in particular is lower than standard.

2.2 Coverage Report

As part of our work assisting DML 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, High: Potential Lockout

DmlContribution

Explanation

The function `changeWalletAddress` is used to set the `dmlwallet` variable, which is used in the initialized modifier. If `changeWalletAddress` is set to 0x0, this will lock out the ability to use `halt/unHalt/changeWalletAddress`. If `changeWalletAddress` is modified to use `onlyOwner` along with `halt/unHalt`, it would allow the wallet address to be set to 0x0 in emergency circumstances, thus avoiding the potential lockout case.

Resolution

The DML Team edited the `changeWalletAddress` function to use `onlyOwner` avoiding the lockout case.

3.2. Resolved, Medium: No Ability To Fix Whitelist

DmlContribution

Explanation

The system is designed in such a way that the whitelist is a single write entity. Because of this, if there is a mistake made in the setting of addresses in the whitelist, it is not possible to fix the incorrect entry.

Resolution

Acknowledged by DML Team.

3.3. Resolved, Medium: Potential Race Condition

DmlContribution

Explanation

The ERC-20 implementation allows for the approval amount to be set again after it is initially set for an address. This creates a potential race condition where a new approved value can be set while there is still an existing approval amount.

Resolution

Acknowledged by DML Team.

3.4. Resolved, Low: Non-Needed Check

StandardToken

Explanation

Within the transferFrom function in this file, there is a check to prevent transfer to the 0 address written as `require(_to != address(0))`. As the same check exists in DmlToken.sol, this becomes unnecessary.

Resolution

Acknowledged by DML Team.

3.5. Resolved, Low: Non-Needed Check

BasicToken

Explanation

Within the transferFrom function in this file, there is a check to prevent transfer to the 0 address written as `require(_to != address(0))`. As the same check exists in DmlToken.sol, this becomes unnecessary.

Resolution

Acknowledged by DML Team.

3.6. Resolved, Low: Two Ownership Systems

DmlContribution

Explanation

The DmlContribution contract utilizes both `onlyWallet` and `onlyOwner`. Compared to `onlyOwner`, `onlyWallet` lacks protection against the 0x0 address and the wallet receiving the ETH and the wallet that is controlling the contract should be the same. Given that, `onlyWallet` should be removed from the contract, and `onlyOwner` should be used in its place.

Resolution

The DML Team has updated this contract to use `onlyOwner`.

3.7. Resolved, Low: Non-Needed Check

DmlContribution

Explanation

In the function `isContract if (_addr == 0) return false;`, `msg.sender` can not be 0x0 as there is a previous check to ensure this.

Resolution

The DML Team has removed the unnecessary check.

3.8. Resolved, Low: No Token Escape

DmlToken

Explanation

The Hosho team suggests adding an escape function for trapped tokens that are not issued by the contract. There are an increasing number of ERC-20 and ERC-223 tokens, some of very large value, getting trapped forever in contracts, so it is valuable to have a function that can return these tokens to a contract issuer or owner for refund.

Resolution

Acknowledged by DML Team.

3.9. Resolved, Low: Missing Event

DmlToken

Explanation

Per the ERC-20 standards, token issuance should fire a `Transfer()` event.

Resolution

Acknowledged by DML Team.

3.10. Resolved, Informational: Transfer Lockout

DmlToken

Explanation

The contract contains functionality to unlock all user tokens through a functional transfer. As the transfers spends a high amount of gas, the Hosho Team recommends implementing a feature similar to the halting modifier contained in the DmlContribution system which would allow the DML team to block all transfers until the crowdsale is over. This provides equivalent protections as the current system while utilizing significantly less gas.

Resolution

Acknowledged by DML Team.

3.11. Resolved, Informational: Non-Needed Modifier

DmlToken

Explanation

The modifier `isLaterThan` is implemented but never used throughout the contract.

Resolution

Removed by the DML Team.

3.12. Resolved, Informational: Non-Needed Modifier

BasicToken

Explanation

The modifier `onlyPayloadSize` is overridden in StandardToken, and thus never used.

Resolution

Acknowledged by the DML Team.

4. Closing Statement

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

The team of experts at Hosho, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, can say with confidence that the DML 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 DML Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

5. Test Suite Results

Changes made:

DmlContribution - ShowTokenAvailable event provided with a name for the variable

DmlContribution - NowTime event provided with a name for the variables

```
 Contract: Pause Tests for DmlToken
  Deployment
    ✓ Should deploy in an un-paused state
  Pause configuration
    ✓ Should be able to be paused (87ms)
    ✓ Should be able to be unpaused (101ms)
    ✓ Should not be able to be unpaused while unpaused
    ✓ Should not be able to be paused while paused (59ms)

 Contract: ERC-20 Tests for DmlToken
  ✓ Should deploy a token with the proper configuration (54ms)
  ✓ Should allocate tokens per the minting function, and validate balances (353ms)
  ✓ Should transfer tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (70ms)
  ✓ Should not transfer negative token amounts
  ✓ Should not transfer more tokens than you have
  ✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (43ms)
  ✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (43ms)
  ✓ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and
```

```
0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to
0x341106cb00828c87cd3ac0de55eda7255e04933f (148ms)
    ✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative
tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b
    ✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer tokens
from 0x667632a620d245b062c0c83c9749c9bfadf84e3b to 0x0
    ✓ Should not transfer tokens to 0x0
    ✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer more
tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b
    ✓ Should allow an approval to be set, then increased, and decreased (159ms)
    ✓ Should only permit the minter to mint tokens
    ✓ Should not permit minting more tokens than the maxTokenAmount
    ✓ Should not allow the minting of tokens if the endTime is passed (100ms)


  Contract: Ownership Tests for DmlToken
    Deployment
      ✓ Should deploy with the owner being the deployer of the contract
    Transfer
      ✓ Should not allow a non-owner to transfer ownership
      ✓ Should not allow the owner to transfer to 0x0
      ✓ Should allow the owner to transfer ownership (55ms)


  Contract: ICO Tests for DecentralizedML
    ✓ Should not deploy with a wallet address of 0x0 (77ms)
    Administration
      ✓ Should only allow the wallet to change the wallet address
      ✓ Should allow the wallet address to be changed (45ms)
      ✓ Should allow the contract to be halted
      ✓ Should allow the contract to be un-halted (74ms)
      ✓ Should allow the max buy limit to be set by the owner (39ms)
```

✓ Should report if the sale is ended (229ms)

Whitelist

✓ Should allow the whitelisting of multiple users (75ms)

✓ Should allow verification if a user is whitelisted (104ms)

✓ Should return the whitelist quota, index, and level (146ms)

✓ Should return correct price rates for a user in various stages (516ms)

Purchasing

✓ Should not allow a contract to purchase tokens (48ms)

✓ Should not allow purchase before the presale time has begun (288ms)

✓ Should not permit a purchase for 0x0

✓ Should require that a L2 whitelist submits > 1 ETH for tokens (39ms)

✓ Should require that a L1/L0 whitelist submits > .1 eth for tokens (44ms)

✓ Should require that a purchaser is whitelisted (99ms)

✓ Should require that contract not be halted

✓ Should require that contract be initialized (44ms)

✓ Should allow a L2 purchase, refunding extra funds, issuing 4158 tokens per eth during whitelist only (134ms)

✓ Should allow a L1 purchase, issuing 3780 tokens per eth (133ms)

✓ Should not allow purchase of more tokens than permitted by the whitelist quota, during the presale period (184ms)

✓ Should allow a L2 purchase, refunding extra funds, issuing 3780 per eth after the whitelist is over (715ms)

✓ Should not allow a single purchase that exceeds the maxBuyLimit (254ms)

6. All Contract Files Tested

File	Fingerprint (SHA256)
contracts/BasicToken.sol	d1905679f6c34e9d1a6201b55caf136777d7f6813f588e7221e6ce4623e9cdb4
contracts/ConvertLib.sol	e9ac39fe496ec73531a334ea63d1a5b806b5e6e49eeef4bde3a7e5121a5382a1
contracts/DmlContribution.sol	975c4a4a745017b316202ee9c8bf768737ca1944a09a70178e4e7349d7c5822f
contracts/DmlToken.sol	0e65944efe3f6777270c78244b3b6ce769bbbd80d230ec75b50978db8415759f
contracts/ERC20.sol	fd3e13a44a215d8115b7c667df7ded995bf32daccc3835afaf8d28343fd9cf8e
contracts/ERC20Basic.sol	62037621d01d27fd5a9dc54f2479b0f1ddd19b8d8839d941b0bd48610bb9be1d
contracts/Ownable.sol	cef5462c26d118afc196c41243ee4e060e990e06708c563a252e2519986d4b50
contracts/Pausable.sol	e6878fd303c8769bb3c7b58a973eaba44d04fe803d6a7e8a98d11351f234b4fc
contracts/SafeMath.sol	63ecc1e3c4c8528a80ee3d8cf9fb37903f8f9553178bbfa46b204bdaed2db898
contracts/StandardToken.sol	0b1fe17d42b6f224dd00fc1281396dbd3fe7198061617633605edc7eb5ea9561

7. Individual File Coverage Report

File	**% Statements**	**% Branches**	**% Functions**	**% Lines**
contracts/BasicToken.sol	85.71%	50.00%	66.67%	75.00%
contracts/ConvertLib.sol	0.00%	100.00%	0.00%	0.00%
contracts/DmlContribution.sol	97.56%	82.81%	100.00%	98.02%
contracts/DmlToken.sol	100.00%	87.50%	100.00%	100.00%
contracts/ERC20.sol	100.00%	100.00%	100.00%	100.00%
contracts/ERC20Basic.sol	100.00%	100.00%	100.00%	100.00%
contracts/Ownable.sol	100.00%	100.00%	100.00%	100.00%
contracts/Pausable.sol	100.00%	100.00%	100.00%	100.00%
contracts/SafeMath.sol	37.50%	30.00%	37.50%	37.50%
contracts/StandardToken.sol	100.00%	83.33%	100.00%	100.00%
All files	**90.85%**	**78.00%**	**88.33%**	**91.67%**