Debaseonomics - SP2

Security Code Review

https://twitter.com/VidarTheAuditor - 1 February 2021

Overview

Project Summary

Project Name	Debaseonomics
Description	Debaseonomics is a combination of Debase, a flexible supply token, working together with Degov, a governance token working together to solve issues faced by similarly designed tokens. 100% of the tokens are distributed through staking and "stabilizer pools" to promote fairness and decentralization.
Platform	Ethereum, Solidity
Codebase	https://github.com/debaseonomics/stabilizers/tree/main/contracts/Debt-Issuance-Pool
Commits	commit b92ede5a96c21881b694dd5d6a42c5572a135dfa

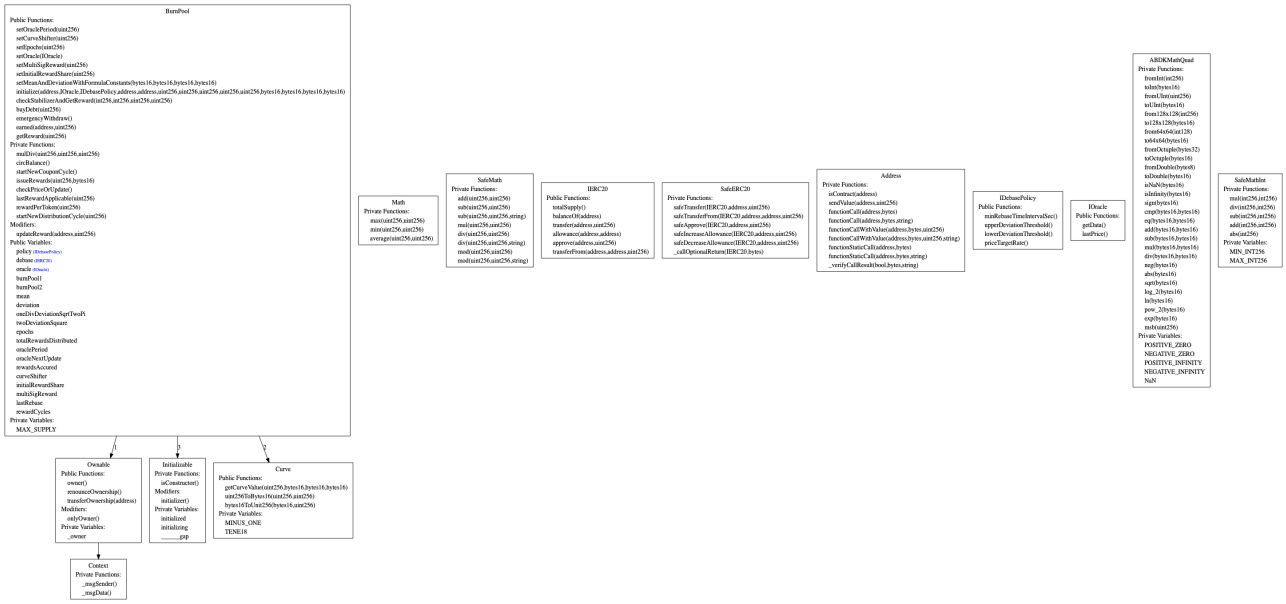
Executive Summary

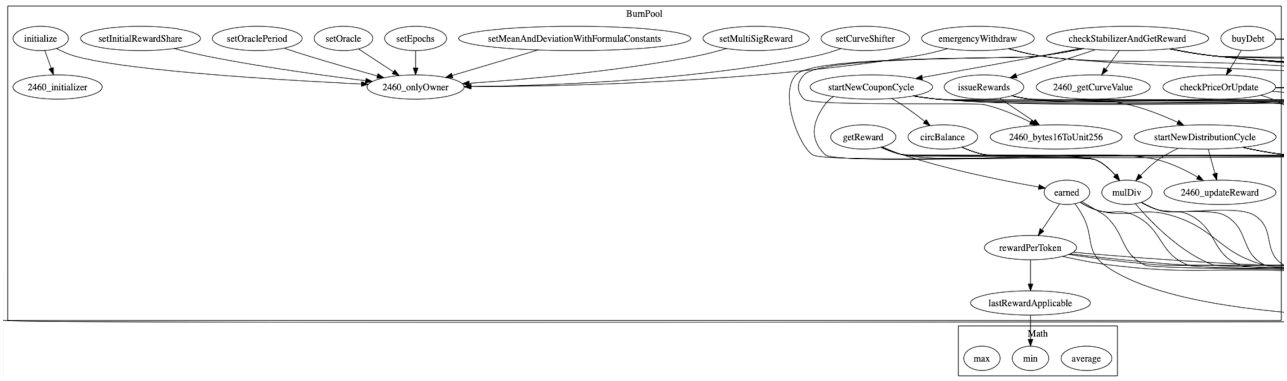
The codebase was found well defined, has proper access restrictions where needed, includes very good comments throughout a code. We have run extensive static analysis of the codebase as well as standard security assessment utilising industry approved tools.

We have found no significant issues during our review.

Architecture & Standards

Architecture of the pool is shown below.





Findings

Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of assembly lines: 0

Use: Openzeppelin-Ownable, Openzeppelin-SafeMath

Name	# functions	ERCS	ERC20 info	Complex code	Features
IDebasePolicy	4			No	
IOracle	2			No	
BurnPool	33			No	Send ETH Tokens interaction Assembly Upgradeable
ABDKMathQuad	30			Yes	
SafeMathInt	6			No	

Static Analysis Findings

High issues: None

Medium issues:

Divide before multiply:

```
BurnPool.mulDiv(uint256,uint256,uint256) (contracts/Debt-Issuance-Pool/BurnPool.sol#135-149) performs a multiplication on the result of a division:
        -a = x.div(z) (contracts/Debt-Issuance-Pool/BurnPool.sol#140)
        -res1 = a.mul(b).mul(z).add(a).mul(d) (contracts/Debt-Issuance-Pool/BurnPool.sol#145)
BurnPool.mulDiv(uint256,uint256,uint256) (contracts/Debt-Issuance-Pool/BurnPool.sol#135-149) performs a multiplication on the result of a division:
        -c = y.div(z) (contracts/Debt-Issuance-Pool/BurnPool.sol#142)
        -res2 = b.mul(c).add(b.mul(d)).div(z) (contracts/Debt-Issuance-Pool/BurnPool.sol#146)
Curve.getCurveValue(uint256,bytes16,bytes16,bytes16) (contracts/Debt-Issuance-Pool/Curve.sol#10-32) performs a multiplication on the result of a division:
        -res2 = ABDKMathQuad.mul(MINUS_ONE,ABDKMathQuad.div(ABDKMathQuad.mul(res1,res1),twoDeviationSquare_)) (contracts/Debt-Issuance-Pool/Curve.sol#21-28)
```

In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

[Manual Check] It does not possesses significant risks for the contract.

Low/Informational issues

state variables that could be declared constant:

```
Curve.MINUS_ONE (contracts/Debt-Issuance-Pool/Curve.sol#7) should be constant
Curve.TENE18 (contracts/Debt-Issuance-Pool/Curve.sol#8) should be constant
```

Dynamic Tests

We have run fuzzing/property-based testing of Ethereum smarts contracts. It was using sophisticated grammar-based fuzzing campaigns based on a contract ABI to falsify user-defined predicates or Solidity assertions.

We found no high level issues.

Manual Check

```
function setMeanAndDeviationWithFormulaConstants(
    bytes16 mean_,
    bytes16 deviation_,
    bytes16 oneDivDeviationSqrtTwoPi_,
    bytes16 twoDeviationSquare_
) external onlyOwner {
    mean = mean_;
    deviation = deviation_;
    oneDivDeviationSqrtTwoPi = oneDivDeviationSqrtTwoPi_;
    twoDeviationSquare = twoDeviationSquare_;

    emit LogSetMeanAndDeviationWithFormulaConstants(
        mean,
        deviation,
        oneDivDeviationSqrtTwoPi,
        twoDeviationSquare
    );
}
```

As this function modified crucial parameters, make sure the contract is deployed with the current governance structure.

Automatic Tests

We have checked the comprehensive test scripts. They validate the functionality of the contracts.

```
Debt Issuance Pool
  Deploy and Initialize
    Oracle initialized settings check
      ✓ Debase address should be correct
      ✓ Dai address should be correct
      ✓ Pool address should be correct
    Burn pool initialization
      ✓ Debase address should be correct
      ✓ Policy address should be correct
      ✓ Oracle address should be correct
      ✓ Burn pool 1 address should be correct
      ✓ Burn pool 2 address should be correct
      ✓ Epochs should be correct
      ✓ Oracle period should be set correctly
      ✓ Curve shifter should be set correctly
      ✓ Oracle period should be set correctly
      ✓ Curve shifter should be set correctly
      ✓ Mean should be set correctly
      ✓ Deviation should be set correctly
      ✓ One dic deviation sqrt two pi should be set correctly
      ✓ Two deviation square should be set correctly
      Basic Functionality
        When first rebase has not fired
          ✓ Users should be not able to buy coupons (45ms)
        When first rebase has been fired
          For neutral supply delta rebase
            ✓ Stabilizer function should not emit an accrue event
            ✓ Reward accrue amount should be zero
            ✓ Reward enum should be set to neutral
            ✓ Users should not be able to buy coupons
          For negative supply delta rebase
            ✓ Stabilizer function should emit new coupon cycle event (128ms)
            ✓ User should be able to buy coupons and emit correct transfer event (63ms)
            ✓ User should emit update oracle event (82ms)
            ✓ User should not be able to buy debt when price goes higher than lower price limit (294ms)
            ✓ User should be able buy debt when price goes back down the limit (326ms)
          For positive supply delta rebase
            ✓ Stabilizer function should emit an accrue event with correct reward amount (356ms)
            ✓ There should be no reward cycles started
            ✓ Another Stabilizer function should emit an accrue event with correct reward amount (126ms)
            ✓ When negative rebase happens new coupon cycle should be started with the correct data (372ms)
            ✓ There should be a reward cycles started afterwards
            When the next rebase is neutral happens followed by a negative
              ✓ The enum should be set to neutral before negative rebase
              ✓ On negative rebase it should start a new coupon cycle with the correct arguments (115ms)
              ✓ Reward cycles should be set to 2
            When next rebase goes from negative to a positive
              When no coupons are bought
                ✓ On positive rebase no rewards distribution cycle should start (313ms)
              When coupons are bought
                ✓ On positive rebase rewards distribution cycle should start with the correct args (527ms)
                ✓ Epoch rewarded should be set to 1
                ✓ Rewards should be earnable
                ✓ Rewards should be claimable (205ms)
                Multisig reward
                  ✓ Multisig claim should be correct
                  ✓ Multisig should get the correct reward amount
                When next rebase is positive
                  ✓ Should start new distribution cycle (202ms)
                  When next rebase is positive again
                    ✓ Should not start new distribution cycle since epoch target is hit (91ms)
                  When next rebase is neutral
                    ✓ No distribution cycle should start (47ms)
                    ✓ Rewards should still be earnable
                  When next rebase is negative
                    ✓ New reward cycle should start (305ms)
                    ✓ Rewards from previous cycle should still be earnable


48 passing (8s)
```

All testes passed successfully.

Deployment & Contract Ownership

The contracts are not deployed yet.

They should be deployed within current security context including multi-sig wallet and governance structure.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, the author assume no responsibility for errors, omissions, or for damages resulting from the use of the provided information. We do not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One review on its own is not enough for a project to be considered secure; that state can only be earned through extensive peer review and extensive testing over an extended period of time.

The information appearing in this report is for general discussion purposes only and is not intended to provide legal security guarantees to any individual or entity.

THIS INFORMATION IS PROVIDED BY "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE HOST OF THIS PROJECT OR ANY CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. WE DO NOT ENDORSE ANY OF THE PROJECTS REVIEWED. THIS IS NOT INVESTMENT ADVICE.