Debaseonomics - incentivizers

Security Code Review

https://twitter.com/VidarTheAuditor - 5 January 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/incentivizers
Commits	commit 03da13d68de62e62cea6075e80a674123f5b79ef

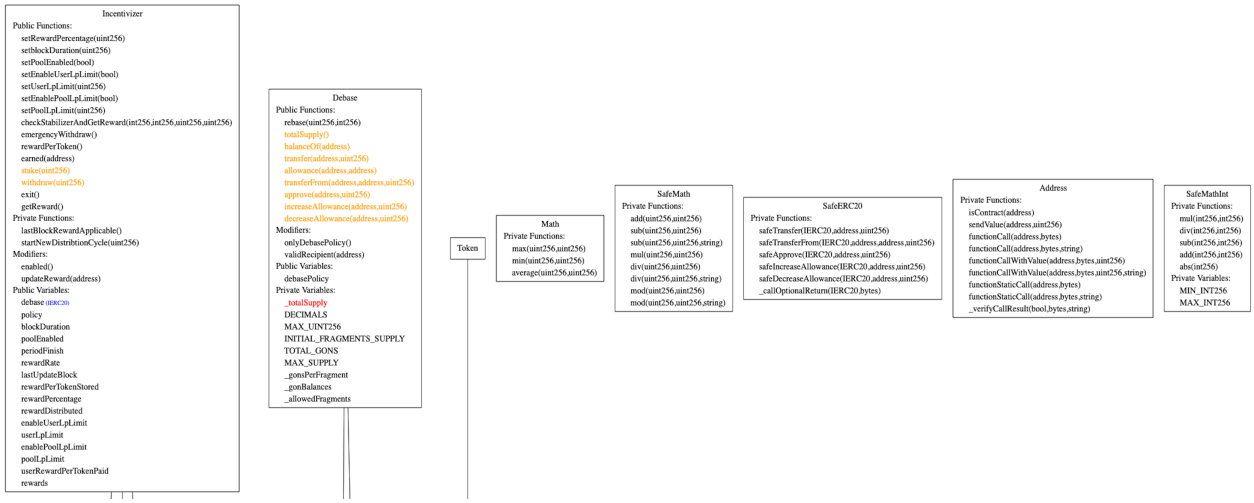
Executive Summary

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 stabiliser pool is shown below. It uses the same pool concept as other Debase pools contracts.

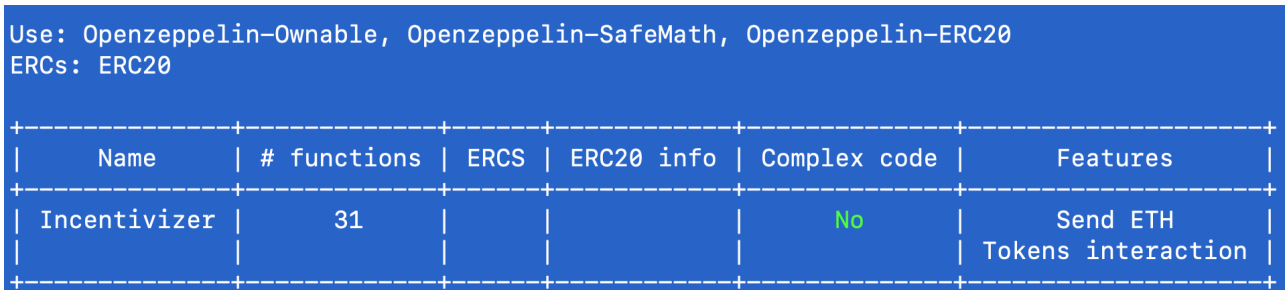


Findings

Number of contracts: 2 (+ 0 in dependencies, + 0 tests)

Number of assembly lines: 0

Use: Openzeppelin-Ownable, Openzeppelin-SafeMath



Static Analysis Findings

High issues:

Reentrancy:

```
Reentrancy in Incentivizer.exit() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#335-338):
        External calls:
        - withdraw(balanceOf(msg.sender)) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#336)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/E
RC20/SafeERC20.sol#69)
                - y.safeTransfer(msg.sender,amount) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#77)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/E
RC20/SafeERC20.sol#69)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
                - debase.safeTransfer(msg.sender,rewardToClaim) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#348)
        External calls sending eth:
        - withdraw(balanceOf(msg.sender)) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#336)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        State variables written after the call(s):
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - _status = _ENTERED (node_modules/@openzeppelin/contracts/utils/ReentrancyGuard.sol#54)
                - _status = _NOT_ENTERED (node_modules/@openzeppelin/contracts/utils/ReentrancyGuard.sol#60)
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - lastUpdateBlock = lastBlockRewardApplicable() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#137)
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - rewardPerTokenStored = rewardPerToken() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#136)
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - rewards[msg.sender] = 0 (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#343)
                - rewards[account] = earned(account) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#139)
        - getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#337)
                - userRewardPerTokenPaid[account] = rewardPerTokenStored (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#140)
```

[Manual Check] As the getReward() function already have nonReentrant check, the exit() function should be protected as well.

After manually checking, it does not posses any risks based on our analysis. Not an issue.

Medium issues:

Divide before multiply:

```
Incentivizer.startNewDistribtionCycle(uint256) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#355-377) performs a multiplication on the res
ult of a division:
        -rewardRate = poolTotalShare.div(blockDuration) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#362)
        -leftover = remaining.mul(rewardRate) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#365)
Debase.constructor() (contracts/Degov-Eth-Incentivizer/Mock/Debase.sol#76-88) performs a multiplication on the result of a division:
        -_gonsPerFragment = TOTAL_GONS.div(_totalSupply) (contracts/Degov-Eth-Incentivizer/Mock/Debase.sol#78)
        -debaseDaiPoolGons = debaseDaiPoolVal.mul(_gonsPerFragment) (contracts/Degov-Eth-Incentivizer/Mock/Debase.sol#83)
```

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

Reentrancy:

```
Reentrancy in Incentivizer.getReward() (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#340-353):
        External calls:
        - debase.safeTransfer(msg.sender,rewardToClaim) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#348)
        State variables written after the call(s):
        - rewardDistributed = rewardDistributed.add(reward) (contracts/Degov-Eth-Incentivizer/Incentivizer.sol#351)
```

[Manual Check] As the getReward() function already have nonReentrant check, the exit() function should be protected as well.

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.

Automatic Tests

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

```
Degov/Eth Incentivizer
  Deploy and Initialize
    Initial settings check
      ✓ Reward token should be debase
      ✓ Pair token should be degov lp
      ✓ Policy should be policy contract
      ✓ Duration should be correct
      ✓ eward Percentage should be correct
      ✓ Pool should be  disabled
      ✓ User lp limit should be enabled
      ✓ User lp limit should be correct
      ✓ Pool lp limit should be enabled
      ✓ Pool lp limit should be correct
  Basic Operation
    User/Pool Lp Limits
      ✓ User cant stake more than user lp limit once (40ms)
      ✓ User cant stake more than user lp limit when combined with previous user stakes (425ms)
      ✓ Users cant stake more pool lp limit
    When Pool is disabled
      ✓ Should not be able to stake
      ✓ Should not be able to withdraw
    When Pool is enabled
      When pool is not rewarded balance
        ✓ Should be enabled
        ✓ Should be able to stake (205ms)
        ✓ Should have correct stake balance
        ✓ Should be able to withdraw (182ms)
        ✓ Should not earn rewards
      When pool is rewarded balance
        For a single user
          Simple Usage
            ✓ Should claim reward with correct amount (50ms)
            ✓ Its reward Rate should be correct (41ms)
            ✓ Should be able to stake (194ms)
            ✓ Should be able to withdraw (209ms)
            ✓ Should earn rewards
            ✓ Should emit a transfer event when rewards are claimed (285ms)
          Claiming Maximum Balance
            ✓ Should have claimable % equal to % of debase sent after reward period has elapsed
            ✓ Should transfer correct amount of debase on get reward (242ms)
  Operation Under Rebases
    Simple Operation
      Positive Rebase
        ✓ Its reward Rate should not change after rebase
        ✓ Should be able to stake (184ms)
        ✓ Should be able to withdraw (179ms)
        ✓ Should earn rewards
        ✓ Should emit a transfer event when rewards are claimed (198ms)
      Negative Rebase
        ✓ Its reward Rate should not change after rebase
        ✓ Should be able to stake (234ms)
        ✓ Should be able to withdraw (182ms)
        ✓ Should earn rewards
        ✓ Should emit a transfer event when rewards are claimed (202ms)
    Claim maximum Balance
      Positive Rebase
        ✓ Should have claimable % equal to % of debase sent after reward period has elapsed
        ✓ Pool balance should be zero after get reward (271ms)
      Negative Rebase
        ✓ Should have claimable % equal to % of debase sent after reward period has elapsed
        ✓ Pool balance should be zero after get reward (247ms)


42 passing (8s)
```

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.