# Debaseonomics - DaiLpPool

## Security Code Review

https://twitter.com/VidarTheAuditor - 19 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/debase-dai-pool |
| **Commits** | commit 7536a339c7004e487acf96d1409b05b14f1c0f29 |

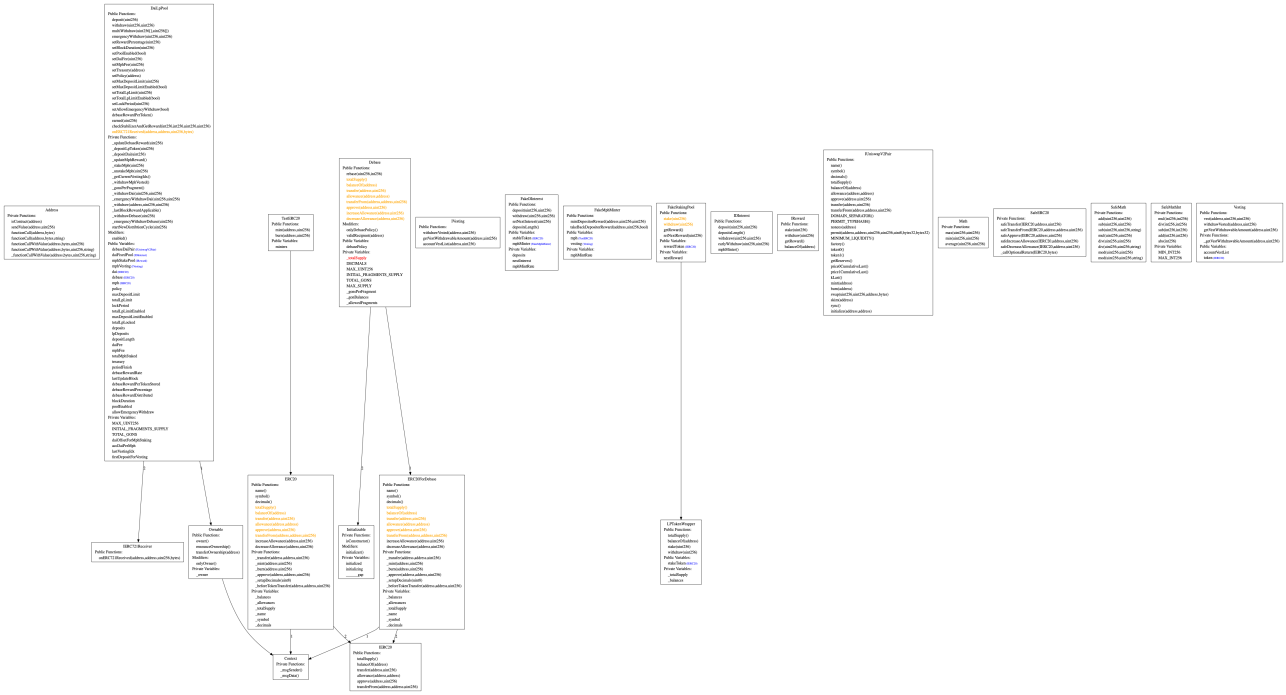
# Executive Summary

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: 5 (+ 0 in dependencies, + 0 tests)

Number of assembly lines: 0

Use: Openzeppelin-Ownable, Openzeppelin-SafeMath

| Name | # functions | ERCS | ERC20 info | Complex code | Features |
|---|---|---|---|---|---|
| IVesting | 3 | | | No | AbiEncoderV2 |
| DaiLpPool | 47 | | | No | Send ETH<br>Tokens interaction<br>AbiEncoderV2 |
| IDInterest | 5 | | | No | |
| IReward | 4 | | | No | |
| IUniswapV2Pair | 27 | ERC20 | ∞ Minting<br>Approve Race Cond. | No | |

# Static Analysis Findings

### High issues: None

### Medium issues:

Divide before multiply:

```
DaiLpPool.startNewDistribtionCycle(uint256) (DaiLpPool.sol#498-519) performs a multiplication on the result of a division:
        -debaseRewardRate = poolTotalShare.div(blockDuration) (DaiLpPool.sol#504)
        -leftover = remaining.mul(debaseRewardRate) (DaiLpPool.sol#507)
```

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

Uses timestamp for comparisons:

```
DaiLpPool._getCurrentVestingIdx() (DaiLpPool.sol#197-205) uses timestamp for comparisons
        Dangerous comparisons:
        - vest.creationTimestamp < block.timestamp (DaiLpPool.sol#200)
DaiLpPool._withdrawMphVested() (DaiLpPool.sol#207-223) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= deposits[depositId].maturationTimestamp (DaiLpPool.sol#216)
DaiLpPool._withdraw(address,uint256,uint256) (DaiLpPool.sol#304-321) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(depositInfo.maturationTimestamp <= block.timestamp,still locked) (DaiLpPool.sol#312)
FakeDInterest.withdraw(uint256,uint256) (mock/FakeDInterest.sol#56-70) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(info.maturationTimestamp <= now,DInterest: Deposit not mature) (mock/FakeDInterest.sol#59)
Vesting.withdrawVested(address,uint256) (mock/Vesting.sol#47-65) uses timestamp for comparisons
        Dangerous comparisons:
        - withdrawnAmount == 0 (mock/Vesting.sol#53)
Vesting._getVestWithdrawableAmount(address,uint256) (mock/Vesting.sol#75-100) uses timestamp for comparisons
        Dangerous comparisons:
        - now >= vestCreationTimestamp.add(vestPeriodInSeconds) (mock/Vesting.sol#88)
```

Manual Check] It does not possesses significant risks for the contract as the pool rewards are based on timestamp.

## 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

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. All run successfully.

```
Contract: DaiLpPool basic testing
  Initial settings check
    ✓ Pair token should be debase-dai uni v2 lp (42ms)
    ✓ Dai fixed pool should be 88mph dai fixed pool
    ✓ Mph staking pool should be 88mph staking pool
    ✓ Mph vesting should be 88mph vesting
    ✓ Check dai token
    ✓ Check debase token (38ms)
    ✓ Check mph token (40ms)
    ✓ Policy should be policy contract
    ✓ Max deposit limit should be zero (134ms)
    ✓ Total lp limit should be zero (131ms)
    ✓ Check lock period
    ✓ TotalLpLimitEnabled should be false (138ms)
    ✓ MaxDepositLimitEnabled should be false
    ✓ TotalLpLocked should be zero (253ms)
    ✓ DepositLength should be zero (132ms)
    ✓ Check tresury
    ✓ Check periodFinish (221ms)
    ✓ Check debaseRewardRate (136ms)
    ✓ Check lastUpdateBlock (126ms)
    ✓ Check debaseRewardPerTokenStored (128ms)
    ✓ Check debaseRewardPercentage
    ✓ Check debaseRewardDistributed (127ms)
    ✓ Check blockDuration
    ✓ Check poolEnabled (125ms)
    ✓ Check allowEmergencyWithdraw
  setters check
    ✓ check setRewardPercentage (146ms)
    ✓ check setBlockDuration (185ms)
    ✓ check setPoolEnabled (208ms)
    ✓ check setDaiFee (114ms)
    ✓ check setMphFee (110ms)
    ✓ check setTreasury (159ms)
    ✓ check setPolicy (133ms)
    ✓ check setMaxDepositLimit (98ms)
    ✓ check setMaxDepositLimitEnabled (141ms)
    ✓ check setTotalLpLimit (92ms)
    ✓ check setTotalLpLimitEnabled (141ms)
    ✓ check setMaxDepositLimit (126ms)
    ✓ check setAllowEmergencyWithdraw (137ms)

Contract: DaiLpPool Mainnet testing
  Test limitations
    ✓ enable/disable pools (7079ms)
    ✓ enable/disable total lp limit (238ms)
    ✓ enable/disable total lp limit (249ms)
  Test single deposit
    ✓ Deposit (10765ms)

Contract: DaiLpPool Mainnet testing
  1) "before all" hook: Get contract references for "Check deposit"


42 passing (49s)
```

## 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.