Debaseonomics

Security Code Review v2

https://twitter.com/VidarTheAuditor - 12 November 2020

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/
Commits	7f8f835e1594e42c00dfb719894ad03ee149abc5

Executive Summary

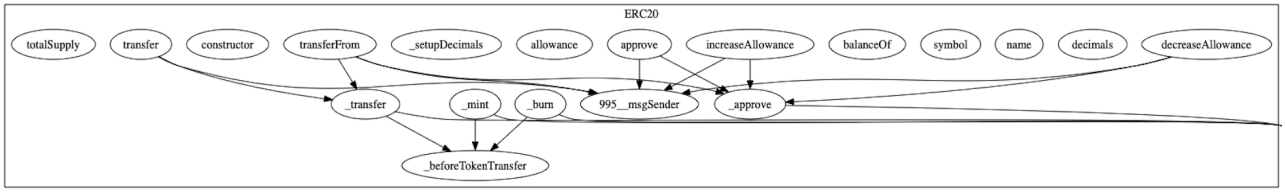
The codebase was found well defined, has proper access restrictions where needed, includes very good comments throughout a code. We have run extensive static analysis of the codebase as well as standard security assessment utilising industry approved tools. We have found some issues during our review that are described later in the report, however overall impression of the code is very good. Project is using SafeMath which is a recommended and good practise. The high complexity of the code can poses some risks (the full functional tests were not conducted). The governance structure used satisfies our tests and current deployment and ownership structure is prepared for the governance implementation. All critical rights are within multi-sig wallet for the interim period before full governance will be in place.

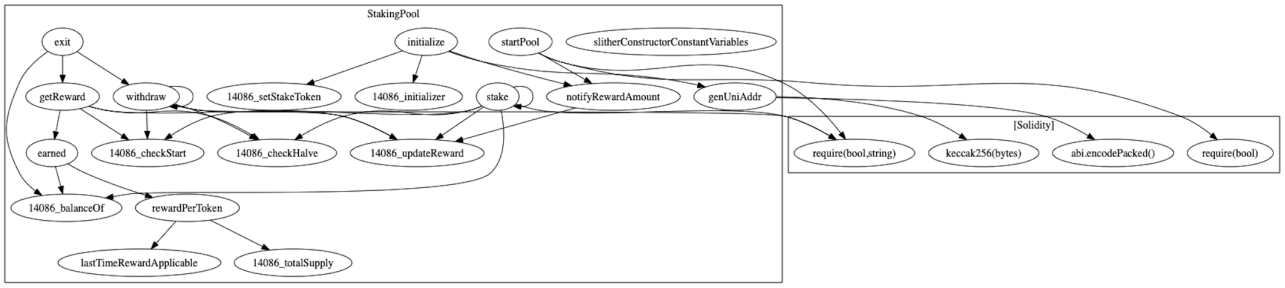
As it is an updated version of the report, all the suggested fixes in the V1 were implemented.

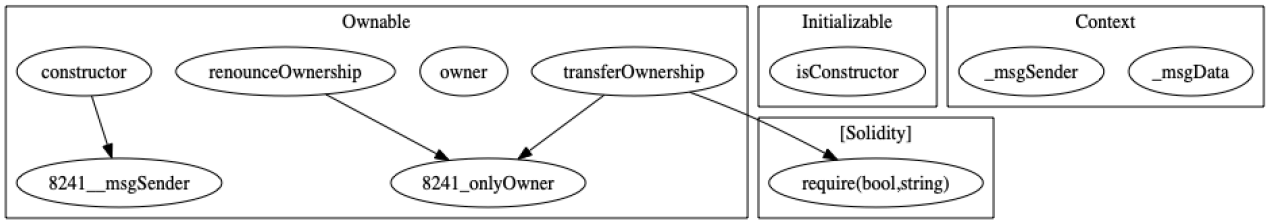
Architecture & Standards

Contracts ecosystem is complex, the calls graph presented below shows the interaction of Debase, StakingPool and DebasePolicy. Those are more important elements of the ecosystem. DebasePolicy initiates a new rebase operation, provided the minimum time period has elapsed. That's the main functionality of the reviewed system.

The complexity of the ecosystem may posses risks.







Debase token is fully ERC-20 compliant.

```
# Check Debase

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] Debase has increaseAllowance(address,uint256)
```

Degov (Compond based governance token) is fully ERC-20 compliant

```
# Check Degov

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] Degov has increaseAllowance(address,uint256)
```

Findings

Number of contracts: 36 (+ 0 in dependencies, + 0 tests)	
Number of lines: 8528 (+ 0 in dependencies, + 0 in tests)	
Number of assembly lines: 0	
Use: SafeMath	

Name	# functions	ERCS	ERC20 info	Complex code	Features
SafeMath	8			No	
Math	3			No	
Address	7			No	Send ETH Assembly
SafeERC20	6			No	Send ETH Tokens interaction
StakingPool	20			No	Send ETH Tokens interaction Assembly Upgradeable
IUniswapV2Factory	8			No	
IUniswapV2Pair	27	ERC20	∞ Minting Approve Race Cond.	No	
Babylonian	1			No	
FixedPoint	10			No	
UniswapV2OracleLibrary	2			No	Tokens interaction
UniswapV2Library	8			No	Tokens interaction
Oracle	13			No	Tokens interaction
Timelock	7			No	Assembly Upgradeable
SafeMathInt	6			No	
Debase	28	ERC20	No Minting Approve Race Cond.	No	Assembly Upgradeable
PoolI	3			No	
DebaseI	4			No	
DebasePolicyI	1			No	
UniV2PairI	1			No	
Orchestrator	7			No	Assembly Upgradeable
UInt256Lib	2			No	AbiEncoderV2
IOracle	1			No	AbiEncoderV2
StabilizerI	2			No	AbiEncoderV2
DebasePolicy	7			No	AbiEncoderV2 Assembly Upgradeable
Degov	32	ERC20	No Minting Approve Race Cond.	No	AbiEncoderV2 Assembly Upgradeable
DegovI	4			No	AbiEncoderV2
GovernorAlpha	7			No	AbiEncoderV2 Assembly Upgradeable
StabilizerPool	8			No	Assembly Upgradeable

Static Analysis Findings

High issues

Reentrancy vulnerabilities:

```
Reentrancy in StakingPool.exit() (StakingPool.sol#943-946):
        External calls:
        - withdraw(balanceOf(msg.sender)) (StakingPool.sol#944)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (StakingPool.sol#580-583)
                - y.safeTransfer(msg.sender,amount) (StakingPool.sol#742)
                - (success,returndata) = target.call{value: weiValue}(data) (StakingPool.sol#448-450)
        - getReward() (StakingPool.sol#945)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (StakingPool.sol#580-583)
                - (success,returndata) = target.call{value: weiValue}(data) (StakingPool.sol#448-450)
                - rewardToken.safeTransfer(msg.sender,reward) (StakingPool.sol#952)
        External calls sending eth:
        - withdraw(balanceOf(msg.sender)) (StakingPool.sol#944)
                - (success,returndata) = target.call{value: weiValue}(data) (StakingPool.sol#448-450)
        - getReward() (StakingPool.sol#945)
                - (success,returndata) = target.call{value: weiValue}(data) (StakingPool.sol#448-450)
        State variables written after the call(s):
        - getReward() (StakingPool.sol#945)
                - initReward = initReward.mul(50).div(100) (StakingPool.sol#783)
        - getReward() (StakingPool.sol#945)
                - lastUpdateTime = lastTimeRewardApplicable() (StakingPool.sol#806)
        - getReward() (StakingPool.sol#945)
                - periodFinish = block.timestamp.add(duration) (StakingPool.sol#786)
        - getReward() (StakingPool.sol#945)
                - rewardPerTokenStored = rewardPerToken() (StakingPool.sol#805)
        - getReward() (StakingPool.sol#945)
                - rewardRate = initReward.div(duration) (StakingPool.sol#785)
        - getReward() (StakingPool.sol#945)
                - rewards[msg.sender] = 0 (StakingPool.sol#951)
                - rewards[account] = earned(account) (StakingPool.sol#808)
        - getReward() (StakingPool.sol#945)
                - userRewardPerTokenPaid[account] = rewardPerTokenStored (StakingPool.sol#809)
```

Manual check: Not an issue after reviewing the code manually.

Exploit Scenario:

```
    function withdrawBalance(){
        // send userBalance[msg.sender] Ether to msg.sender
        // if mgs.sender is a contract, it will call its fallback function
        if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
            throw;
        }
        userBalance[msg.sender] = 0;
    }
```

Bob uses the re-entrancy bug to call `withdrawBalance` two times, and withdraw more than its initial deposit to the contract.

Possible Solution:
https://solidity.readthedocs.io/en/v0.4.21/security-considerations.html# re-entrancy

Medium issues:

Divide before multiply:

```
StakingPool.notifyRewardAmount(uint256) (StakingPool.sol#958-974) performs a multiplication on the result of a division:
        -rewardRate = reward.div(duration) (StakingPool.sol#964)
        -leftover = remaining.mul(rewardRate) (StakingPool.sol#967)
```

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.

Low/Informational issues

Boolean equality:

```
StakingPool.startPool() (StakingPool.sol#873-881) compares to a boolean constant:
        -require(bool,string)(poolStarted == false,Pool can only be started once) (StakingPool.sol#875)
StakingPool.checkStart() (StakingPool.sol#792-802) compares to a boolean constant:
        -require(bool,string)(poolStarted == true,Orchestrator hasn't started pool) (StakingPool.sol#794)
```

Manual check: Boolean constants can be used directly and do not need to be compare to true or false.

Public functions that could be declared external:

```
setRewardDistribution(address) should be declared external:
        - IRewardDistributionRecipient.setRewardDistribution(address) (StakingPool.sol#707-709)
initialize(string,address,address,bool,address,uint256,bool,uint256,uint256,bool,uint256) should be declared external:
        - StakingPool.initialize(string,address,address,bool,address,uint256,bool,uint256,uint256,bool,uint256) (StakingPool.sol#837-871)
```

Public functions that are never called by the contract should be declared external to save gas.

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 following high level issues.

```
==== Integer Arithmetic Bugs ====
SWC ID: 101
Severity: High
Contract: StakingPool
Function name: poolName()
PC address: 7023
Estimated Gas Usage: 3282 - 4600
The arithmetic operator can underflow.
It is possible to cause an integer overflow or underflow in the arithmetic operation.
--------------------
In file: /tmp/StakingPool.sol:751

string public poolName

--------------------
Initial State:

Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [SOMEGUY], function: initialize(string,address,address,bool,address,uint256,bool,uint256,uint256,bool,uint256), txdata: 0xe72739f5000000000000
0000000000000000000000000000000000000000000000000000000108000000000000000000000000000000000000000000000000000000000000003700000000000000000000000000000000
0000000000000000000000000000000101010101010101010101010101010101010101010101010101010101010000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
01010101010101010101010101000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [SOMEGUY], function: poolName(), txdata: 0xf3466dfa, value: 0x0
```

```
==== Integer Arithmetic Bugs ====
SWC ID: 101
Severity: High
Contract: StakingPool
Function name: poolName()
PC address: 7075
Estimated Gas Usage: 3282 - 4600
The arithmetic operator can underflow.
It is possible to cause an integer overflow or underflow in the arithmetic operation.
--------------------
In file: /tmp/StakingPool.sol:751

string public poolName

--------------------
Initial State:

Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [SOMEGUY], function: initialize(string,address,address,bool,address,uint256,bool,uint256,uint256,bool,uint256), txdata: 0xe72739f5000000000000
0000000000000000000000000000000000000000000000000000000108000000000000000000000000000000000000000000000000000000000000003700000000000000000000000000000000
0000000000000000000000000000000101010101010101010101010101010101010101010101010101010101010000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
01010101010101010101010101000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [SOMEGUY], function: poolName(), txdata: 0xf3466dfa, value: 0x0
```

Manual check: The variable poolName is used only in initialise function once so that vulnerability most probably does not posses such a high risk.

The tests were successfully passed. The failed tests on the diagram above are false positive as the fuzzer did not run any transactions for them.

Automatic Tests

The project lacks any automated tests.

Deployment & Contract Ownership

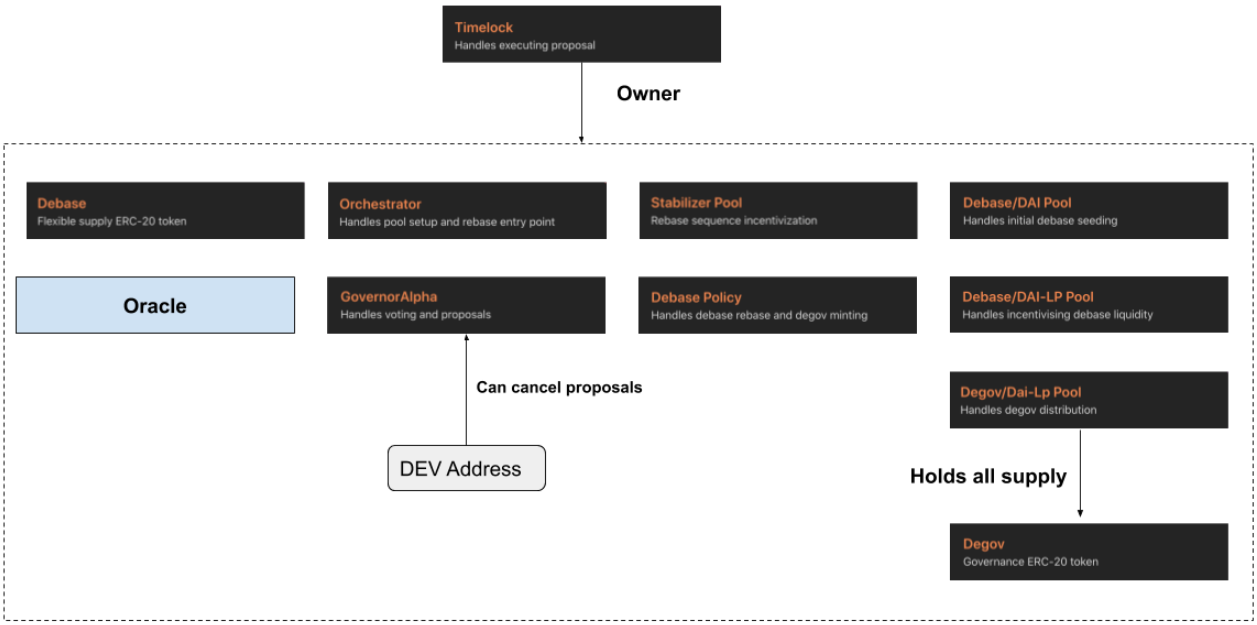
The contracts are deployed to the following addresses:

"degov": "0x469E66e06fEc34839E5eB1273ba85A119B8D702F",
"debase": "0x9248c485b0B80f76DA451f167A8db30F33C70907",
"debasePolicy": "0x989Edd2e87B1706AB25b2E8d9D9480DE3Cc383eD",
"governorAlpha": "0x291BC8eDFE98155224502282444cC2E98d80d2d5",
"timelock": "0x969e1d56682305963c6b7f8920D0200189B22482",
"debaseDaiPool": "0xf5cB771023706Ca566eA6128b88e03A262737479",
"debaseDaiLpPool": "0xF4168cc431e9a8310e595dB9F7E2564cC96F5D51",
"degovDaiLpPool": "0xaB68de2a9d9A733F3c4CFE52Af7Fc4f6aa015637",
"orchestrator": "0x177A1F55Df0F28d8e9F5C837C706E04A82890025",
"dai": "0x6b175474e89094c44da98b954eedeac495271d0f",
"debaseDaiLp": "0xE98f89a2B3AeCDBE2118202826478Eb02434459A",
"oracle": "0xb1Df2F0C76074eD466510F4440772Cc7b3D5337C",
"stabilizerPool": "0x99d6EB950F9719d7b883a2c67735ecA6A91d6EaD"

The proper governance structure has to be established by moving ownership of critical contracts to Timelock contract. Timelock contract is part of Compound like governance DAO (https://compound.finance/docs/governance).

However, the interim check has been implemented by moving all ownership to multi-sig wallet.

The desired snapshot of ownership is show below:



When all parameters are double checked after rebase is executed, the aforementioned structure should be implemented.

Overview of issues found

- Issues presented in the sections above connected to static analysis **(not considered substantial as of now)**
- Source code available at Github it is not the same as the one verified on etherescan and deployed
- Not verified Oracle contract
  - **[Fixed] After talking to DEV**
- Dev has still access to cancelling function (via **guardian** property which is set to DEV address) **[Fixed] All ownership currently in multi-sig wallet.**
- **[Fixed by new deployment and airdrop]** ~~During the discussion with DEV, it was found that **minRebaseTimeIntervalSec** parameter of the DebasePolicy contract is set incorrectly. It was verified to be true. The code deployed has the following:~~

```
minRebaseTimeIntervalSec = 5 minutes;
```

  - ~~It means that in the current setup, rebases would happen very 5 minutes, which is not as specified in the requirements (24 h).~~
  - ~~**That issue is a substantial bug as it would mean irrational rebasing behaviour may happen. As the governance structure is already in place changing that parameter requires farming for the requires Degov tokens. However farming can be start ONLY after rebase is turned on. It is somewhat catch 22 situation that need further study in order to establish possible actions to fix it.**~~

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.