0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Champion Finance

July 2022

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for Champion Finance.

The Champion Finance Protocol allows users to farm ShareTokens and MainTokens. The ShareToken is an ERC20-like token with initial distribution for GenesisRewardPool and ShareTokenRewardPool farms. The MainToken is a rebase token and can be farmed in the NodePool contract.

Contracts Treasury and Boardroom allow keeping a stable price of the MainToken using rebase mechanism.

The code is available at the GitHub repository and was audited after the commit 8b709fcfafc540566084d6daf8c29e63fd78c4ad. The contracts' code was updated by the Champion Finance team (commit 4b584f24fc12aad0bc9220fcca6baa60364b0aab) according to this report.

The inspected contracts are: Boardroom.sol, MainToken.sol, ShareToken.sol, Treasury.sol, GenesisRewardPool.sol, NodePool.sol, ShareTokenRewardPool.sol. The inspected contracts (not MainToken and ShareToken) use upgradeable deployment scheme. Users have to trust the owner, who can change the contracts' logic at their will.

Name	Champion Finance
Audit date	2022-07-11 - 2022-07-20
Language	Solidity
Platform	Avalanche Network

Contracts checked

Name	Address
ShareToken	
MainToken	

ContractGuard

Boardroom

Treasury

GenesisRewardPool

NodePool

ShareTokenRewardPool

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed

Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Reentrancy - FIXED (ContractGuard)

In the current version, the `onlyOneBlock()` modifier does not fully protect the contract from a reentrancy attack.

```
    modifier onlyOneBlock() {
        require(!checkSameOriginReentranted(), "ContractGuard: one block, one
function");
        require(!checkSameSenderReentranted(), "ContractGuard: one block, one
function");

        _;

        _status[block.number][tx.origin] = true;
        _status[block.number][msg.sender] = true;
    }
```

Recommendation: We recommend replacing function execution at the end of modifier `onlyOneBlock()` in the following way:

```
    modifier onlyOneBlock() {
        require(!checkSameOriginReentranted(), "ContractGuard: one block, one
function");
        require(!checkSameSenderReentranted(), "ContractGuard: one block, one
function");

        _status[block.number][tx.origin] = true;
        _status[block.number][msg.sender] = true;

        _;
    }
```

It will prevent re-rentrancy attacks in other contracts.

2. The issue of centralization - FIXED (Treasury)

The function `allocateSeigniorage()` calls the `_updatePrice()` function inside. And the `_updatePrice()` function is restricted by the `onlyOperator` modifier:

```
function allocateSeigniorage() external onlyOneBlock checkCondition checkEpoch
checkOperator {
    _updatePrice();
    ...
 }

function _updatePrice() internal onlyOperator {
    try IOracle(oracle).update() {} catch {
    revert("Treasury: failed to update price from the oracle");
 }
```

Only the `allocateSeigniorage()` function allows changing the `epoch`, which is important for contract `Boardroom`.

If the private key of the operator (owner) of the `Treasury` contract is lost (or the operator is gone), then the users who staked the tokens in the `Boardroom` contract will not be able to withdraw them until the epoch is changed.

Recommendation: Restrictions on calling function `_updatePrice()` by ordinary users should be removed.

3. Changing oracle - FIXED (Treasury)

The contract operator can change the implementation of the oracle using the `setOracle()` function. This can lead to a completely different calculation of the weighted token price.

```
    function setOracle(address _oracle) external onlyOperator {
        oracle = _oracle;
    }
```

Recommendation: Restrict operator (owner) ability to change oracle contract address.

4. Possible locked rewards - FIXED (GenesisRewardPool)

Rewards can become partially locked if new pools are added after the `poolStartTime` timestamp without calling the `massUpdatePools()` function inside the `add()` function.

Recommendation: We recommend using a hard-coded flag `_withUpdate = true` inside the `add()` function.

5. Possible locked rewards - FIXED (NodePool)

Rewards can become partially locked if new pools are added after the `poolStartTime` timestamp without calling `massUpdatePools()` function inside the `add()` function.

Recommendation: We recommend using a hard-coded flag `_withUpdate = true` inside the `add()` function.

6. Possible locked rewards - FIXED (ShareTokenRewardPool)

Rewards can become partially locked if new pools are added after the `poolStartTime` timestamp without calling the `massUpdatePools()` function inside the add() function.

Recommendation: We recommend using hard-coded flag `_withUpdate = true` inside the `add()` function.

7. Unreachable dev reward - FIXED (ShareTokenRewardPool)

The variable `_daoFundReward` must be replaced with `_devFundReward` in L292.

Recommendation: We recommend fixing this typo, otherwise, dev funds will be locked.

Medium severity issues

1. Block gas limit in rebase - FIXED (MainToken)

The `maxExclusion` variable allows excluding from rebase up to 50000 address.

The `rebase()` function calls the functions `_burnExcludedAccountTokens()`, `_mintExcludedAccountTokens()` which update arrays of 50000 elements in a loop. Since the block gas limit is about 8M gas in the Avalanche C-chain, the call of the function `rebase()` with 50000 excluded addresses will fail.

Recommendation: Reduce the limit of addresses that can be excluded from rebasing.

2. Reentrancy problem - FIXED (GenesisRewardPool)

Reentrancy is possible if the pool's token address refers to an ERC20 token with transfer hooks. In that case, any user can claim a reward multiple times as `user.rewardDebt` is updated after the token transfer.

Recommendation: The contract owner must avoid adding pools with non-standard staking tokens.

3. Reentrancy problem - FIXED (NodePool)

Reentrancy is possible if the pool's token address refers to an ERC20 token with transfer hooks. In that case, any user can claim a reward multiple times as `user.rewardDebt` is updated after the token transfer.

Recommendation: The contract owner must avoid adding pools with non-standard staking tokens.

4. Reentrancy problem - FIXED (ShareTokenRewardPool)

Reentrancy is possible if the pool's token address refers to an ERC20 token with transfer hooks. In that case, any user can claim a reward multiple times as `user.rewardDebt` is updated after the token transfer.

Recommendation: The contract owner must avoid adding pools with non-standard staking tokens.

Low severity issues

1. Gas optimization - FIXED (ShareToken)

The library `SafeMath` is never used in the contract and can be removed to save gas on deployment.

2. Function call order (ShareToken)

The Treasury contract is expected to be the ShareToken operator. But the Treasury contract hasn't the functionality to call the `distributeReward()` function. Thus, the `distributeReward()` function must be called before the transfer of operator rights.

Recommendation: Consider replacing the call of the `distributeReward()` function into the contract constructor.

3. Typos - FIXED (MainToken)

Typos in the code impair its readability:- 'FUNTION' in L219, L327.

4. Unprotected mint by the owner - FIXED (MainToken)

The owner of the contract has the ability to change the operator of the contract to a controlled address at any time. Thus, the owner will be able to mint an unlimited number of tokens. It can lead to an unfair tokenomic.

Recommendation: It is important to restrict the owner's (operator's) right to mint tokens, e.g. by transferring ownership to a Timelock contract.

5. Few events - FIXED (MainToken)

Many functions from the contract lack events:

1. `_transferOperator()`
2. `disableRebase()`
3. `enableRebase()`

6. Gas optimization - FIXED (MainToken)

1. The library `SafeMath8` is never used in the contract and can be removed to save gas on deployment.

2. The state variable `MAX_UINT256` is never used and can be removed to save gas on deployment.

3. The value of the `excluded.length` in L153, L166, L182, L208 can be stored in the local variable to save gas.

4. Visibility of the functions `__Upgradeable_Init()`, `operator()`, `isOperator()`, `transferOperator()`, `grantRebaseExclusion()` can be declared as '`external`' to save gas.

7. Lack of validation - FIXED (Boardroom)

We recommend adding non-zero validation for address parameters of the `initialize()` function.

8. Few events - FIXED (Boardroom)

Most contract functions do not emit events. Consider adding events to the governance functions.

9. Double initialization (Boardroom)

The order of the contract initialization can be broken in the following way.

1. Deployer calls the `initialize()` function and changes the flag `initialized` to the 'true' value.

2. Deployer (or an arbitrary user) calls the `__Upgradeable_Init()` function and it changes the flag `initialized` back to the 'false' value.

3. Now any user has the ability to call the `initialize()` function again with other arguments (other token addresses).

Recommendation: It is necessary to rigidly fix the stages of the contract initialization.

10. Gas optimization - PARTIALLY FIXED (Boardroom)

1. Values `lastSnapshotIndex` and `epochTimerStart` of the `Memberseat` structure can be cast to `uint128` type and packed into one slot to save gas.

2. Visibility of the functions `__Upgradeable_Init()`, `initialize()`, `rewardPerShare()` can be declared as '`external`' to save gas.

11. Few events - FIXED (Treasury)

Most contract functions do not emit events. Consider adding events to the 'setter' functions.

12. Lack of validation - FIXED (Treasury)

We recommend adding validation for parameters of the `initialize()` function.

13. Double initialization (Treasury)

The order of the contract initialization can be broken in the following way.

1. Deployer calls the `initialize()` function and it changes the flag `initialized` to the '`true`' value.

2. Deployer calls the `__Upgradeable_Init()` function and it changes the flag `initialized` back to the '`false`' value.

3. Now any user has the ability to call the `initialize()` function again with other arguments (other token addresses).

Recommendation: It is necessary to rigidly fix the stages of contract initialization.

14. Gas optimization - FIXED (Treasury)

1. Using the `if` statements in L279, L284, L296, L304 are redundant because the `daoFundSharedPercent, devFundSharedPercent` variables are constant and more than zero.

2. The variable `lastRebaseTimestampSec` is never used and can be removed.

3. Visibility of the functions `__Upgradeable_Init()`, `initialize()`, `isInitialized()`, `getTwapPrice()`, `getEstimatedReward()` can be declared as '`external`' to save gas.

4. The functionality of the `getMainTokenCirculatingSupply()` function is similar to the `MainToken.rebaseSupply()` function. Using a direct call to `MainToken.rebaseSupply()` will allow saving gas.

5. We recommend using require instead of assert statements. This will save gas on execution and make it easier to track errors.

15. Gas optimization - FIXED (GenesisRewardPool)

1. Visibility of the functions `__Upgradeable_Init()`, `deposit()`, `withdraw()` can be declared as '`external`' to save gas.

2. The variables `TOTAL_REWARDS`, `TOTAL_USER_REWARDS`, `DAO_FUND_REWARDS` are never used and can be removed.

16. Lack of validation - FIXED (GenesisRewardPool)

We recommend adding non-zero validation for address parameters of the `__Upgradeable_Init()` function.

17. Pending rewards in view function - FIXED (GenesisRewardPool)

The `deposit()` and `withdraw()` functions take part of the reward (`airdropTip`) for the `airdropWallet` (L243, L285). But the `pending()` function doesn't take this into account. Thus, this will lead to the fact that users expected one number of rewards, but received another.

Recommendation: We recommend matching the function `pending()` with functions `deposit()` and `withdraw()`.

18. Lack of validation - FIXED (NodePool)

We recommend adding validation for parameters of the functions `__Upgradeable_Init()`, `add()`.

19. Gas optimization - FIXED (NodePool)

1. Visibility of the functions `__Upgradeable_Init()`, `set()`, `deposit()`, `getMainTokenCirculatingSupply()`, `withdraw()` can be declared as '`external`' to save gas.

2. The variables `TOTAL_USER_REWARDS`, `DAO_FUND_REWARDS`, `DAO_FUND_REWARDS` are never used and can be removed.

3. The structures `UserTicket` and `DevFundInfo` are never used and can be removed.

4. The functionality of the `getMainTokenCirculatingSupply()` function is similar to `MainToken.rebaseSupply()` function. Using a direct call to `MainToken.rebaseSupply()` will allow saving gas.

20. Typos - FIXED (ShareTokenRewardPool)

There is a typo in the code comment in L62: there should be '650000' instead of '700000'.

21. Lack of validation - FIXED (ShareTokenRewardPool)

We recommend adding validation for parameters of the functions `__Upgradeable_Init()`, `add()`.

22. Gas optimization - FIXED (ShareTokenRewardPool)

1. Visibility of the functions `__Upgradeable_Init()`, `set()`, `deposit()`, `withdraw()` can be declared as '`external`' to save gas.

2. The variables `TOTAL_USER_REWARDS`, `DAO_FUND_REWARDS`, `DEV_FUND_REWARDS` are never used and can be removed.

Conclusion

Champion Finance ShareToken, MainToken, ContractGuard, Boardroom, Treasury, GenesisRewardPool, NodePool, ShareTokenRewardPool contracts were audited. 7 high, 4 medium, 22 low severity issues were found.

According to this report 7 high, 4 medium, 19 low issues were fixed by the Champion Finance team.

Most of the contracts are upgradeable (not MainToken and ShareToken), they're going to be deployed via proxies. Users interacting with the contracts have to trust the owner. MainToken and ShareToken contract's ownership was renounced.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Slither output

```
Treasury._sendToBoardroom(uint256) (Treasury.sol#291-317) ignores return value by
mainTokenErc20.transfer(daoFund,_daoFundSharedAmount) (Treasury.sol#299)
Treasury._sendToBoardroom(uint256) (Treasury.sol#291-317) ignores return value by
mainTokenErc20.transfer(devFund,_devFundSharedAmount) (Treasury.sol#307)
EmergencyWithdraw.emergencyWithdrawTokenBalance(address,address,uint256) (utils/
EmergencyWithdraw.sol#45-52) ignores return value by erc20.transfer(_to,_amount) (utils/
EmergencyWithdraw.sol#51)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-
transfer

NodePool.pendingShare(uint256,address) (distribution/NodePool.sol#204-236) performs a
multiplication on the result of a division:
        -_shareTokenReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(distribution/NodePool.sol#219-221)
        -accShareTokenPerShare =
accShareTokenPerShare.add(_shareTokenReward.mul(1e18).div(totalTokenStaked))
(distribution/NodePool.sol#224-226)
NodePool.updatePool(uint256) (distribution/NodePool.sol#247-280) performs a
multiplication on the result of a division:
        -_shareTokenReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(distribution/NodePool.sol#271-273)
        -pool.accShareTokenPerShare =
pool.accShareTokenPerShare.add(_shareTokenReward.mul(1e18).div(totalTokenStaked))
(distribution/NodePool.sol#275-277)
GenesisRewardPool.pending(uint256,address) (distribution/GenesisRewardPool.sol#165-180)
performs a multiplication on the result of a division:
        -_mainTokenReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(distribution/GenesisRewardPool.sol#172)
        -accMainTokenPerShare =
accMainTokenPerShare.add(_mainTokenReward.mul(1e18).div(tokenSupply)) (distribution/
GenesisRewardPool.sol#173)
GenesisRewardPool.updatePool(uint256) (distribution/GenesisRewardPool.sol#208-228)
performs a multiplication on the result of a division:
        -_mainTokenReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(distribution/GenesisRewardPool.sol#224)
        -pool.accMainTokenPerShare =
pool.accMainTokenPerShare.add(_mainTokenReward.mul(1e18).div(tokenSupply))
```

```
(distribution/GenesisRewardPool.sol#225)
Treasury.getEstimatedReward() (Treasury.sol#270-289) performs a multiplication on the
result of a division:
        -estimatedReward = mainTokenTotalSupply.mul(percentage).div(10000)
(Treasury.sol#276)
        -_daoFundSharedAmount = estimatedReward.mul(daoFundSharedPercent).div(100)
(Treasury.sol#280)
Treasury.getEstimatedReward() (Treasury.sol#270-289) performs a multiplication on the
result of a division:
        -estimatedReward = mainTokenTotalSupply.mul(percentage).div(10000)
(Treasury.sol#276)
        -_devFundSharedAmount = estimatedReward.mul(devFundSharedPercent).div(100)
(Treasury.sol#285)
Treasury.computeSupplyDelta() (Treasury.sol#385-398) performs a multiplication on the
result of a division:
        -rebasePercentage = rate.sub(targetRate).mul(ONE).div(targetRate)
(Treasury.sol#394)
        -supplyDelta =
mathRound(getMainTokenCirculatingSupply().mul(rebasePercentage).div(ONE))
(Treasury.sol#397)
Treasury.mathRound(uint256) (Treasury.sol#400-408) performs a multiplication on the
result of a division:
        -valueFloor = _value.div(midpointRounding).mul(midpointRounding)
(Treasury.sol#401)
ShareTokenRewardPool.pendingShare(uint256,address) (distribution/
ShareTokenRewardPool.sol#171-187) performs a multiplication on the result of a
division:
        -_shareTokenReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(distribution/ShareTokenRewardPool.sol#178)
        -accShareTokenPerShare =
accShareTokenPerShare.add(_shareTokenReward.mul(1e18).div(tokenSupply)) (distribution/
ShareTokenRewardPool.sol#179)
ShareTokenRewardPool.updatePool(uint256) (distribution/
ShareTokenRewardPool.sol#232-252) performs a multiplication on the result of a
division:
        -_shareTokenReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint)
(distribution/ShareTokenRewardPool.sol#248)
        -pool.accShareTokenPerShare =
pool.accShareTokenPerShare.add(_shareTokenReward.mul(1e18).div(tokenSupply))
(distribution/ShareTokenRewardPool.sol#249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-
multiply
```

```
NodePool.checkPoolDuplicate(IERC20Upgradeable,uint256) (distribution/
NodePool.sol#109-115) uses a dangerous strict equality:
        - isPoolDuplicate = poolInfo[pid].lockTime == _lockTime && poolInfo[pid].token
== _token (distribution/NodePool.sol#112)
GenesisRewardPool.updatePool(uint256) (distribution/GenesisRewardPool.sol#208-228) uses
a dangerous strict equality:
        - tokenSupply == 0 (distribution/GenesisRewardPool.sol#214)
ShareTokenRewardPool.updatePool(uint256) (distribution/
ShareTokenRewardPool.sol#232-252) uses a dangerous strict equality:
        - tokenSupply == 0 (distribution/ShareTokenRewardPool.sol#238)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-
strict-equalities

Reentrancy in Treasury._rebase(uint256) (Treasury.sol#410-428):
        External calls:
        - newTotalSupply = IMainToken(mainToken).rebase(epoch,supplyDelta,negative)
(Treasury.sol#422)
        State variables written after the call(s):
        - previousEpoch = epoch (Treasury.sol#424)
Reentrancy in Treasury.initialize(address,address,address,address,uint256)
(Treasury.sol#174-202):
        External calls:
        - IMainToken(mainToken).grantRebaseExclusion(address(this)) (Treasury.sol#195)
        - IMainToken(mainToken).grantRebaseExclusion(address(boardroom))
(Treasury.sol#196)
        State variables written after the call(s):
        - initialized = true (Treasury.sol#198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-1

Treasury.setSupplyTiersEntry(uint8,uint256) (Treasury.sol#225-236) contains a tautology
or contradiction:
        - require(bool,string)(_index >= 0,Index has to be higher than 0)
(Treasury.sol#226)
Treasury.setMaxExpansionTiersEntry(uint8,uint256) (Treasury.sol#238-244) contains a
tautology or contradiction:
        - require(bool,string)(_index >= 0,Index has to be higher than 0)
(Treasury.sol#239)
Treasury.calculateMaxSupplyExpansionPercent(uint256) (Treasury.sol#319-333) contains a
tautology or contradiction:
```

```
        - tierId >= 0 (Treasury.sol#325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-
contradiction

Treasury.getTwapPrice().price (Treasury.sol#165) is a local variable never initialized
Treasury.getMainTokenPrice().price (Treasury.sol#157) is a local variable never
initialized
Treasury.calculateMaxSupplyExpansionPercent(uint256).maxSupplyExpansionPercent
(Treasury.sol#320) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-
local-variables

Treasury.getMainTokenPrice() (Treasury.sol#156-162) ignores return value by
IOracle(oracle).consult(mainToken,1e18) (Treasury.sol#157-161)
Treasury.getTwapPrice() (Treasury.sol#164-170) ignores return value by
IOracle(oracle).twap(mainToken,1e18) (Treasury.sol#165-169)
Treasury._sendToBoardroom(uint256) (Treasury.sol#291-317) ignores return value by
mainTokenErc20.mint(address(this),_amount) (Treasury.sol#293)
Treasury.allocateSeigniorage() (Treasury.sol#335-379) ignores return value by
IMainToken(mainToken).mint(daoFund,_savedForBoardroom) (Treasury.sol#354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Treasury.setOperator(address) (Treasury.sol#204-206) should emit an event for:
        - operator = _operator (Treasury.sol#205)
Treasury.setBoardroom(address) (Treasury.sol#208-210) should emit an event for:
        - boardroom = _boardroom (Treasury.sol#209)
MainToken.__Upgradeable_Init(string,string) (MainToken.sol#47-57) should emit an event
for:
        - _operator = _msgSender() (MainToken.sol#53)
Boardroom.setOperator(address) (Boardroom.sol#142-144) should emit an event for:
        - operator = _operator (Boardroom.sol#143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-
access-control

NodePool.add(uint256,IERC20Upgradeable,bool,uint256,uint256) (distribution/
NodePool.sol#118-160) should emit an event for:
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (distribution/
NodePool.sol#158)
NodePool.set(uint256,uint256) (distribution/NodePool.sol#163-172) should emit an event
for:
        - totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(_allocPoint)
```

```
(distribution/NodePool.sol#167-169)
GenesisRewardPool.add(uint256,IERC20Upgradeable,bool,uint256) (distribution/
GenesisRewardPool.sol#107-137) should emit an event for:
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (distribution/
GenesisRewardPool.sol#135)
GenesisRewardPool.set(uint256,uint256) (distribution/GenesisRewardPool.sol#140-148)
should emit an event for:
        - totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(_allocPoint)
(distribution/GenesisRewardPool.sol#144)
ShareTokenRewardPool.add(uint256,IERC20Upgradeable,bool,uint256) (distribution/
ShareTokenRewardPool.sol#104-142) should emit an event for:
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (distribution/
ShareTokenRewardPool.sol#140)
ShareTokenRewardPool.set(uint256,uint256) (distribution/
ShareTokenRewardPool.sol#145-154) should emit an event for:
        - totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(_allocPoint)
(distribution/ShareTokenRewardPool.sol#149-151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-
arithmetic

GenesisRewardPool.__Upgradeable_Init(address,address,uint256)._airdropWallet
(distribution/GenesisRewardPool.sol#80) lacks a zero-check on :
                - airdropWallet = _airdropWallet (distribution/
GenesisRewardPool.sol#90)
Treasury.initialize(address,address,address,address,uint256)._mainToken
(Treasury.sol#175) lacks a zero-check on :
                - mainToken = _mainToken (Treasury.sol#181)
Treasury.initialize(address,address,address,address,uint256)._shareToken
(Treasury.sol#176) lacks a zero-check on :
                - shareToken = _shareToken (Treasury.sol#182)
Treasury.initialize(address,address,address,address,uint256)._oracle (Treasury.sol#177)
lacks a zero-check on :
                - oracle = _oracle (Treasury.sol#183)
Treasury.initialize(address,address,address,address,uint256)._boardroom
(Treasury.sol#178) lacks a zero-check on :
                - boardroom = _boardroom (Treasury.sol#184)
Treasury.setOperator(address)._operator (Treasury.sol#204) lacks a zero-check on :
                - operator = _operator (Treasury.sol#205)
Treasury.setBoardroom(address)._boardroom (Treasury.sol#208) lacks a zero-check on :
                - boardroom = _boardroom (Treasury.sol#209)
Treasury.setOracle(address)._oracle (Treasury.sol#216) lacks a zero-check on :
```

```
                - oracle = _oracle (Treasury.sol#217)
Boardroom.setOperator(address)._operator (Boardroom.sol#142) lacks a zero-check on :
                - operator = _operator (Boardroom.sol#143)
ShareTokenRewardPool.__Upgradeable_Init(address,address,uint256)._mainToken
(distribution/ShareTokenRewardPool.sol#77) lacks a zero-check on :
                - mainToken = _mainToken (distribution/ShareTokenRewardPool.sol#83)
EmergencyWithdraw.emergencyWithdrawEthBalance(address,uint256)._to (utils/
EmergencyWithdraw.sol#28) lacks a zero-check on :
                - address(_to).transfer(_amount) (utils/EmergencyWithdraw.sol#29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-
address-validation

NodePool.getMainTokenCirculatingSupply() (distribution/NodePool.sol#356-364) has
external calls inside a loop: balanceExcluded =
balanceExcluded.add(mainToken.balanceOf(excludedFromTotalSupply[entryId]))
(distribution/NodePool.sol#361)
GenesisRewardPool.updatePool(uint256) (distribution/GenesisRewardPool.sol#208-228) has
external calls inside a loop: tokenSupply = pool.token.balanceOf(address(this))
(distribution/GenesisRewardPool.sol#213)
Treasury.getMainTokenCirculatingSupply() (Treasury.sol#259-268) has external calls
inside a loop: balanceExcluded =
balanceExcluded.add(mainTokenErc20.balanceOf(excludedFromTotalSupply[entryId]))
(Treasury.sol#265)
ShareTokenRewardPool.updatePool(uint256) (distribution/
ShareTokenRewardPool.sol#232-252) has external calls inside a loop: tokenSupply =
pool.token.balanceOf(address(this)) (distribution/ShareTokenRewardPool.sol#237)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-
a-loop

Variable 'Treasury.getMainTokenPrice().price (Treasury.sol#157)' in
Treasury.getMainTokenPrice() (Treasury.sol#156-162) potentially used before
declaration: uint256(price) (Treasury.sol#158)
Variable 'Treasury.getTwapPrice().price (Treasury.sol#165)' in Treasury.getTwapPrice()
(Treasury.sol#164-170) potentially used before declaration: uint256(price)
(Treasury.sol#166)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-
declaration-usage-of-local-variables

Reentrancy in Treasury.allocateSeigniorage() (Treasury.sol#335-379):
        External calls:
        - _updatePrice() (Treasury.sol#336)
```

```
                - IOracle(oracle).update() (Treasury.sol#254-256)
        State variables written after the call(s):
        - previousEpochMainPrice = getMainTokenPrice() (Treasury.sol#338)
Reentrancy in Treasury.initialize(address,address,address,address,uint256)
(Treasury.sol#174-202):
        External calls:
        - IMainToken(mainToken).grantRebaseExclusion(address(this)) (Treasury.sol#195)
        - IMainToken(mainToken).grantRebaseExclusion(address(boardroom))
(Treasury.sol#196)
        State variables written after the call(s):
        - operator = msg.sender (Treasury.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-2

Reentrancy in Treasury._rebase(uint256) (Treasury.sol#410-428):
        External calls:
        - newTotalSupply = IMainToken(mainToken).rebase(epoch,supplyDelta,negative)
(Treasury.sol#422)
        Event emitted after the call(s):
        - LogRebase(epoch,supplyDelta,targetRate,_oldPrice,newTotalSupply,oldTotalSupply
,block.timestamp) (Treasury.sol#427)
Reentrancy in Treasury._sendToBoardroom(uint256) (Treasury.sol#291-317):
        External calls:
        - mainTokenErc20.mint(address(this),_amount) (Treasury.sol#293)
        - mainTokenErc20.transfer(daoFund,_daoFundSharedAmount) (Treasury.sol#299)
        Event emitted after the call(s):
        - DaoFundFunded(block.timestamp,_daoFundSharedAmount) (Treasury.sol#300)
Reentrancy in Treasury._sendToBoardroom(uint256) (Treasury.sol#291-317):
        External calls:
        - mainTokenErc20.mint(address(this),_amount) (Treasury.sol#293)
        - mainTokenErc20.transfer(daoFund,_daoFundSharedAmount) (Treasury.sol#299)
        - mainTokenErc20.transfer(devFund,_devFundSharedAmount) (Treasury.sol#307)
        Event emitted after the call(s):
        - DevFundFunded(block.timestamp,_devFundSharedAmount) (Treasury.sol#308)
Reentrancy in Treasury._sendToBoardroom(uint256) (Treasury.sol#291-317):
        External calls:
        - mainTokenErc20.mint(address(this),_amount) (Treasury.sol#293)
        - mainTokenErc20.transfer(daoFund,_daoFundSharedAmount) (Treasury.sol#299)
        - mainTokenErc20.transfer(devFund,_devFundSharedAmount) (Treasury.sol#307)
        - IERC20Upgradeable(mainToken).safeApprove(boardroom,0) (Treasury.sol#313)
        - IERC20Upgradeable(mainToken).safeApprove(boardroom,_amount)
(Treasury.sol#314)
```

```
        - IBoardroom(boardroom).allocateSeigniorage(_amount) (Treasury.sol#315)
        Event emitted after the call(s):
        - BoardroomFunded(block.timestamp,_amount) (Treasury.sol#316)
Reentrancy in Boardroom.allocateSeigniorage(uint256) (Boardroom.sol#231-244):
        External calls:
        - mainToken.safeTransferFrom(msg.sender,address(this),amount)
(Boardroom.sol#242)
        Event emitted after the call(s):
        - RewardAdded(msg.sender,amount) (Boardroom.sol#243)
Reentrancy in Boardroom.claimReward() (Boardroom.sol#220-229):
        External calls:
        - mainToken.safeTransfer(msg.sender,reward) (Boardroom.sol#226)
        Event emitted after the call(s):
        - RewardPaid(msg.sender,reward) (Boardroom.sol#227)
Reentrancy in Treasury.initialize(address,address,address,address,uint256)
(Treasury.sol#174-202):
        External calls:
        - IMainToken(mainToken).grantRebaseExclusion(address(this)) (Treasury.sol#195)
        - IMainToken(mainToken).grantRebaseExclusion(address(boardroom))
(Treasury.sol#196)
        Event emitted after the call(s):
        - Initialized(msg.sender,block.number) (Treasury.sol#201)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-3

NodePool.__Upgradeable_Init(address,address,uint256) (distribution/NodePool.sol#88-102)
uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < _poolStartTime,late) (distribution/
NodePool.sol#93)
NodePool.checkPoolDuplicate(IERC20Upgradeable,uint256) (distribution/
NodePool.sol#109-115) uses timestamp for comparisons
        Dangerous comparisons:
        - pid < length (distribution/NodePool.sol#111)
        - isPoolDuplicate = poolInfo[pid].lockTime == _lockTime && poolInfo[pid].token
== _token (distribution/NodePool.sol#112)
        - require(bool,string)(! isPoolDuplicate,NodePool is existed) (distribution/
NodePool.sol#113)
NodePool.add(uint256,IERC20Upgradeable,bool,uint256,uint256) (distribution/
NodePool.sol#118-160) uses timestamp for comparisons
        Dangerous comparisons:
```

```
        - block.timestamp < poolStartTime (distribution/NodePool.sol#129)
        - _lastRewardTime == 0 (distribution/NodePool.sol#131)
        - _lastRewardTime < poolStartTime (distribution/NodePool.sol#134)
        - _lastRewardTime == 0 || _lastRewardTime < block.timestamp (distribution/
NodePool.sol#140)
        - _isStarted = (_lastRewardTime <= poolStartTime) || (_lastRewardTime <=
block.timestamp) (distribution/NodePool.sol#144-145)
NodePool.getGeneratedReward(uint256,uint256) (distribution/NodePool.sol#175-201) uses
timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/NodePool.sol#180)
        - _toTime >= poolEndTime (distribution/NodePool.sol#182)
        - _toTime <= poolStartTime (distribution/NodePool.sol#192)
NodePool.pendingShare(uint256,address) (distribution/NodePool.sol#204-236) uses
timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > pool.lastRewardTime && totalTokenStaked != 0 (distribution/
NodePool.sol#213)
NodePool.updatePool(uint256) (distribution/NodePool.sol#247-280) uses timestamp for
comparisons
        Dangerous comparisons:
        - block.timestamp <= pool.lastRewardTime (distribution/NodePool.sol#249)
NodePool.pendingDaoFund(uint256,uint256,address) (distribution/NodePool.sol#282-297)
uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/NodePool.sol#284)
        - _toTime >= poolEndTime (distribution/NodePool.sol#285)
        - _toTime <= poolStartTime (distribution/NodePool.sol#290)
NodePool.pendingDevFund(uint256,uint256,address) (distribution/NodePool.sol#299-314)
uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/NodePool.sol#301)
        - _toTime >= poolEndTime (distribution/NodePool.sol#302)
        - _toTime <= poolStartTime (distribution/NodePool.sol#307)
NodePool.deposit(uint256,uint256) (distribution/NodePool.sol#317-354) uses timestamp
for comparisons
        Dangerous comparisons:
        - timeDeposit < poolStartTime (distribution/NodePool.sol#328)
NodePool.withdraw(uint256,uint256) (distribution/NodePool.sol#382-428) uses timestamp
for comparisons
        Dangerous comparisons:
```

```
        - require(bool,string)(user.amount >= _amount,withdraw: not good) (distribution/
NodePool.sol#386)
        - timeWithdraw < poolStartTime (distribution/NodePool.sol#390)
        - _daoFundReward > 0 (distribution/NodePool.sol#396)
        - _devFundReward > 0 (distribution/NodePool.sol#402)
        - require(bool,string)(duringTime > pool.lockTime,Not enough time to claim
reward) (distribution/NodePool.sol#411)
        - require(bool,string)(duringTime_scope_0 > pool.lockTime,Not enough time to
withdraw) (distribution/NodePool.sol#418)
NodePool.safeShareTokenTransfer(address,uint256) (distribution/NodePool.sol#431-440)
uses timestamp for comparisons
        Dangerous comparisons:
        - _amount > _shareTokenBalance (distribution/NodePool.sol#434)
GenesisRewardPool.__Upgradeable_Init(address,address,uint256) (distribution/
GenesisRewardPool.sol#78-92) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < _poolStartTime,late) (distribution/
GenesisRewardPool.sol#83)
GenesisRewardPool.checkPoolDuplicate(IERC20Upgradeable) (distribution/
GenesisRewardPool.sol#99-104) uses timestamp for comparisons
        Dangerous comparisons:
        - pid < length (distribution/GenesisRewardPool.sol#101)
        - require(bool,string)(poolInfo[pid].token != _token,GenesisPool: existing
pool?) (distribution/GenesisRewardPool.sol#102)
GenesisRewardPool.add(uint256,IERC20Upgradeable,bool,uint256) (distribution/
GenesisRewardPool.sol#107-137) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < poolStartTime (distribution/GenesisRewardPool.sol#117)
        - _lastRewardTime == 0 (distribution/GenesisRewardPool.sol#119)
        - _lastRewardTime < poolStartTime (distribution/GenesisRewardPool.sol#122)
        - _lastRewardTime == 0 || _lastRewardTime < block.timestamp (distribution/
GenesisRewardPool.sol#128)
        - _isStarted = (_lastRewardTime <= poolStartTime) || (_lastRewardTime <=
block.timestamp) (distribution/GenesisRewardPool.sol#132)
GenesisRewardPool.getGeneratedReward(uint256,uint256) (distribution/
GenesisRewardPool.sol#151-162) uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/GenesisRewardPool.sol#152)
        - _toTime >= poolEndTime (distribution/GenesisRewardPool.sol#153)
        - _toTime <= poolStartTime (distribution/GenesisRewardPool.sol#158)
GenesisRewardPool.pending(uint256,address) (distribution/GenesisRewardPool.sol#165-180)
```

```
uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > pool.lastRewardTime && tokenSupply != 0 (distribution/
GenesisRewardPool.sol#170)
GenesisRewardPool.pendingDaoFund(uint256,uint256,address) (distribution/
GenesisRewardPool.sol#182-197) uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/GenesisRewardPool.sol#184)
        - _toTime >= poolEndTime (distribution/GenesisRewardPool.sol#185)
        - _toTime <= poolStartTime (distribution/GenesisRewardPool.sol#190)
GenesisRewardPool.updatePool(uint256) (distribution/GenesisRewardPool.sol#208-228) uses
timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= pool.lastRewardTime (distribution/
GenesisRewardPool.sol#210)
GenesisRewardPool.withdraw(uint256,uint256) (distribution/
GenesisRewardPool.sol#260-298) uses timestamp for comparisons
        Dangerous comparisons:
        - _daoFundReward > 0 (distribution/GenesisRewardPool.sol#270)
        - _reward > 0 (distribution/GenesisRewardPool.sol#279)
        - airdropTip > 0 (distribution/GenesisRewardPool.sol#281)
        - userReward > 0 (distribution/GenesisRewardPool.sol#286)
GenesisRewardPool.safeMainTokenTransfer(address,uint256) (distribution/
GenesisRewardPool.sol#301-310) uses timestamp for comparisons
        Dangerous comparisons:
        - _amount > _mainTokenBalance (distribution/GenesisRewardPool.sol#304)
ShareTokenRewardPool.__Upgradeable_Init(address,address,uint256) (distribution/
ShareTokenRewardPool.sol#75-89) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < _poolStartTime,late) (distribution/
ShareTokenRewardPool.sol#80)
ShareTokenRewardPool.checkPoolDuplicate(IERC20Upgradeable) (distribution/
ShareTokenRewardPool.sol#96-101) uses timestamp for comparisons
        Dangerous comparisons:
        - pid < length (distribution/ShareTokenRewardPool.sol#98)
        - require(bool,string)(poolInfo[pid].token != _token,ShareTokenRewardPool:
existing pool?) (distribution/ShareTokenRewardPool.sol#99)
ShareTokenRewardPool.add(uint256,IERC20Upgradeable,bool,uint256) (distribution/
ShareTokenRewardPool.sol#104-142) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < poolStartTime (distribution/ShareTokenRewardPool.sol#114)
```

```
        - _lastRewardTime == 0 (distribution/ShareTokenRewardPool.sol#116)
        - _lastRewardTime < poolStartTime (distribution/ShareTokenRewardPool.sol#119)
        - _lastRewardTime == 0 || _lastRewardTime < block.timestamp (distribution/
ShareTokenRewardPool.sol#125)
        - _isStarted = (_lastRewardTime <= poolStartTime) || (_lastRewardTime <=
block.timestamp) (distribution/ShareTokenRewardPool.sol#129-131)
ShareTokenRewardPool.getGeneratedReward(uint256,uint256) (distribution/
ShareTokenRewardPool.sol#157-168) uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/ShareTokenRewardPool.sol#158)
        - _toTime >= poolEndTime (distribution/ShareTokenRewardPool.sol#159)
        - _toTime <= poolStartTime (distribution/ShareTokenRewardPool.sol#164)
ShareTokenRewardPool.pendingShare(uint256,address) (distribution/
ShareTokenRewardPool.sol#171-187) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > pool.lastRewardTime && tokenSupply != 0 (distribution/
ShareTokenRewardPool.sol#176)
ShareTokenRewardPool.pendingDaoFund(uint256,uint256,address) (distribution/
ShareTokenRewardPool.sol#189-204) uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/ShareTokenRewardPool.sol#191)
        - _toTime >= poolEndTime (distribution/ShareTokenRewardPool.sol#192)
        - _toTime <= poolStartTime (distribution/ShareTokenRewardPool.sol#197)
ShareTokenRewardPool.pendingDevFund(uint256,uint256,address) (distribution/
ShareTokenRewardPool.sol#206-221) uses timestamp for comparisons
        Dangerous comparisons:
        - _fromTime >= _toTime (distribution/ShareTokenRewardPool.sol#208)
        - _toTime >= poolEndTime (distribution/ShareTokenRewardPool.sol#209)
        - _toTime <= poolStartTime (distribution/ShareTokenRewardPool.sol#214)
ShareTokenRewardPool.updatePool(uint256) (distribution/
ShareTokenRewardPool.sol#232-252) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= pool.lastRewardTime (distribution/
ShareTokenRewardPool.sol#234)
ShareTokenRewardPool.withdraw(uint256,uint256) (distribution/
ShareTokenRewardPool.sol#276-313) uses timestamp for comparisons
        Dangerous comparisons:
        - _daoFundReward > 0 (distribution/ShareTokenRewardPool.sol#287)
        - _devFundReward > 0 (distribution/ShareTokenRewardPool.sol#292)
        - _reward > 0 (distribution/ShareTokenRewardPool.sol#302)
ShareTokenRewardPool.safeShareTokenTransfer(address,uint256) (distribution/
```

```
ShareTokenRewardPool.sol#316-325) uses timestamp for comparisons
        Dangerous comparisons:
        - _amount > _shareTokenBalance (distribution/ShareTokenRewardPool.sol#319)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-
timestamp

NodePool.updatePool(uint256) (distribution/NodePool.sol#247-280) has costly operations
inside a loop:
        - totalAllocPoint = totalAllocPoint.add(pool.allocPoint) (distribution/
NodePool.sol#261)
GenesisRewardPool.updatePool(uint256) (distribution/GenesisRewardPool.sol#208-228) has
costly operations inside a loop:
        - totalAllocPoint = totalAllocPoint.add(pool.allocPoint) (distribution/
GenesisRewardPool.sol#220)
MainToken.revokeRebaseExclusion(address) (MainToken.sol#205-217) has costly operations
inside a loop:
        - excluded.pop() (MainToken.sol#212)
ShareTokenRewardPool.updatePool(uint256) (distribution/
ShareTokenRewardPool.sol#232-252) has costly operations inside a loop:
        - totalAllocPoint = totalAllocPoint.add(pool.allocPoint) (distribution/
ShareTokenRewardPool.sol#244)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-
operations-inside-a-loop

Pragma version0.8.13 (distribution/NodePool.sol#3) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (distribution/GenesisRewardPool.sol#3) necessitates a version too
recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (Treasury.sol#3) necessitates a version too recent to be trusted.
Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (owner/Operator.sol#3) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (utils/Epoch.sol#3) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (utils/ContractGuard.sol#3) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (ShareToken.sol#3) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (ERC20TokenDecimal6.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (MainToken.sol#3) necessitates a version too recent to be trusted.
```

```
Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (ERC20Token.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (Boardroom.sol#3) necessitates a version too recent to be trusted.
Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (distribution/ShareTokenRewardPool.sol#3) necessitates a version
too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.13 (utils/EmergencyWithdraw.sol#2) necessitates a version too recent
to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity

Variable NodePool.lastDaoFundRewardTime (distribution/NodePool.sol#74) is too similar
to NodePool.lastDevFundRewardTime (distribution/NodePool.sol#77)
Variable NodePool.shareTokenPerSecondForDaoFund (distribution/NodePool.sol#73) is too
similar to NodePool.shareTokenPerSecondForDevFund (distribution/NodePool.sol#76)
Variable GenesisRewardPool.TOTAL_REWARD_POOL_0_NEXT_PHASE (distribution/
GenesisRewardPool.sol#58) is too similar to
GenesisRewardPool.TOTAL_REWARD_POOL_1_NEXT_PHASE (distribution/
GenesisRewardPool.sol#57)
Variable GenesisRewardPool.TOTAL_REWARD_POOL_0_NEXT_PHASE (distribution/
GenesisRewardPool.sol#58) is too similar to
GenesisRewardPool.TOTAL_REWARD_POOL_2_NEXT_PHASE (distribution/
GenesisRewardPool.sol#56)
Variable GenesisRewardPool.TOTAL_REWARD_POOL_1_NEXT_PHASE (distribution/
GenesisRewardPool.sol#57) is too similar to
GenesisRewardPool.TOTAL_REWARD_POOL_2_NEXT_PHASE (distribution/
GenesisRewardPool.sol#56)
Variable Treasury.maxPercentExpansionTier (Treasury.sol#77) is too similar to
Treasury.minPercentExpansionTier (Treasury.sol#76)
Variable ShareTokenRewardPool.lastDaoFundRewardTime (distribution/
ShareTokenRewardPool.sol#66) is too similar to
ShareTokenRewardPool.lastDevFundRewardTime (distribution/ShareTokenRewardPool.sol#69)
Variable ShareTokenRewardPool.shareTokenPerSecondForDaoFund (distribution/
ShareTokenRewardPool.sol#65) is too similar to
ShareTokenRewardPool.shareTokenPerSecondForDevFund (distribution/
ShareTokenRewardPool.sol#68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-
are-too-similar
```

```
Treasury.initialize(address,address,address,address,uint256) (Treasury.sol#174-202)
uses literals with too many digits:
        - supplyTiers = (0,500000000000000000000000,1000000000000000000000000,1500000000
000000000000000,2000000000000000000000000,5000000000000000000000000,1000000000000000000
000000,2000000000000000000000000,5000000000000000000000000) (Treasury.sol#192)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits

MainToken.MAX_UINT256 (MainToken.sol#26) is never used in MainToken
(MainToken.sol#12-365)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-
variable

Treasury.lastRebaseTimestampSec (Treasury.sol#61) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-constant

__Upgradeable_Init(address,address,uint256) should be declared external:
        - NodePool.__Upgradeable_Init(address,address,uint256) (distribution/
NodePool.sol#88-102)
set(uint256,uint256) should be declared external:
        - NodePool.set(uint256,uint256) (distribution/NodePool.sol#163-172)
deposit(uint256,uint256) should be declared external:
        - NodePool.deposit(uint256,uint256) (distribution/NodePool.sol#317-354)
getMainTokenCirculatingSupply() should be declared external:
        - NodePool.getMainTokenCirculatingSupply() (distribution/NodePool.sol#356-364)
getPoolAvailableStake(uint256) should be declared external:
        - NodePool.getPoolAvailableStake(uint256) (distribution/NodePool.sol#371-379)
withdraw(uint256,uint256) should be declared external:
        - NodePool.withdraw(uint256,uint256) (distribution/NodePool.sol#382-428)
__Upgradeable_Init(address,address,uint256) should be declared external:
        - GenesisRewardPool.__Upgradeable_Init(address,address,uint256) (distribution/
GenesisRewardPool.sol#78-92)
deposit(uint256,uint256) should be declared external:
        - GenesisRewardPool.deposit(uint256,uint256) (distribution/
GenesisRewardPool.sol#231-257)
withdraw(uint256,uint256) should be declared external:
        - GenesisRewardPool.withdraw(uint256,uint256) (distribution/
GenesisRewardPool.sol#260-298)
__Upgradeable_Init() should be declared external:
        - Treasury.__Upgradeable_Init() (Treasury.sol#96-104)
```

```
isInitialized() should be declared external:
        - Treasury.isInitialized() (Treasury.sol#146-148)
getTwapPrice() should be declared external:
        - Treasury.getTwapPrice() (Treasury.sol#164-170)
initialize(address,address,address,address,uint256) should be declared external:
        - Treasury.initialize(address,address,address,address,uint256)
(Treasury.sol#174-202)
getEstimatedReward() should be declared external:
        - Treasury.getEstimatedReward() (Treasury.sol#270-289)
isOperator() should be declared external:
        - Operator.isOperator() (owner/Operator.sol#27-29)
transferOperator(address) should be declared external:
        - Operator.transferOperator(address) (owner/Operator.sol#31-33)
getCurrentEpoch() should be declared external:
        - Epoch.getCurrentEpoch() (utils/Epoch.sol#57-59)
getPeriod() should be declared external:
        - Epoch.getPeriod() (utils/Epoch.sol#61-63)
getStartTime() should be declared external:
        - Epoch.getStartTime() (utils/Epoch.sol#65-67)
getLastEpochTime() should be declared external:
        - Epoch.getLastEpochTime() (utils/Epoch.sol#69-71)
__Upgradeable_Init(string,string) should be declared external:
        - MainToken.__Upgradeable_Init(string,string) (MainToken.sol#47-57)
operator() should be declared external:
        - MainToken.operator() (MainToken.sol#59-61)
isOperator() should be declared external:
        - MainToken.isOperator() (MainToken.sol#68-70)
transferOperator(address) should be declared external:
        - MainToken.transferOperator(address) (MainToken.sol#72-74)
grantRebaseExclusion(address) should be declared external:
        - MainToken.grantRebaseExclusion(address) (MainToken.sol#192-199)
__Upgradeable_Init() should be declared external:
        - Boardroom.__Upgradeable_Init() (Boardroom.sol#117-119)
initialize(IERC20Upgradeable,IERC20Upgradeable,ITreasury) should be declared external:
        - Boardroom.initialize(IERC20Upgradeable,IERC20Upgradeable,ITreasury)
(Boardroom.sol#122-140)
rewardPerShare() should be declared external:
        - Boardroom.rewardPerShare() (Boardroom.sol#188-190)
__Upgradeable_Init(address,address,uint256) should be declared external:
        - ShareTokenRewardPool.__Upgradeable_Init(address,address,uint256)
(distribution/ShareTokenRewardPool.sol#75-89)
```

```
set(uint256,uint256) should be declared external:
        - ShareTokenRewardPool.set(uint256,uint256) (distribution/
ShareTokenRewardPool.sol#145-154)
deposit(uint256,uint256) should be declared external:
        - ShareTokenRewardPool.deposit(uint256,uint256) (distribution/
ShareTokenRewardPool.sol#255-273)
withdraw(uint256,uint256) should be declared external:
        - ShareTokenRewardPool.withdraw(uint256,uint256) (distribution/
ShareTokenRewardPool.sol#276-313)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
```

0x Guard