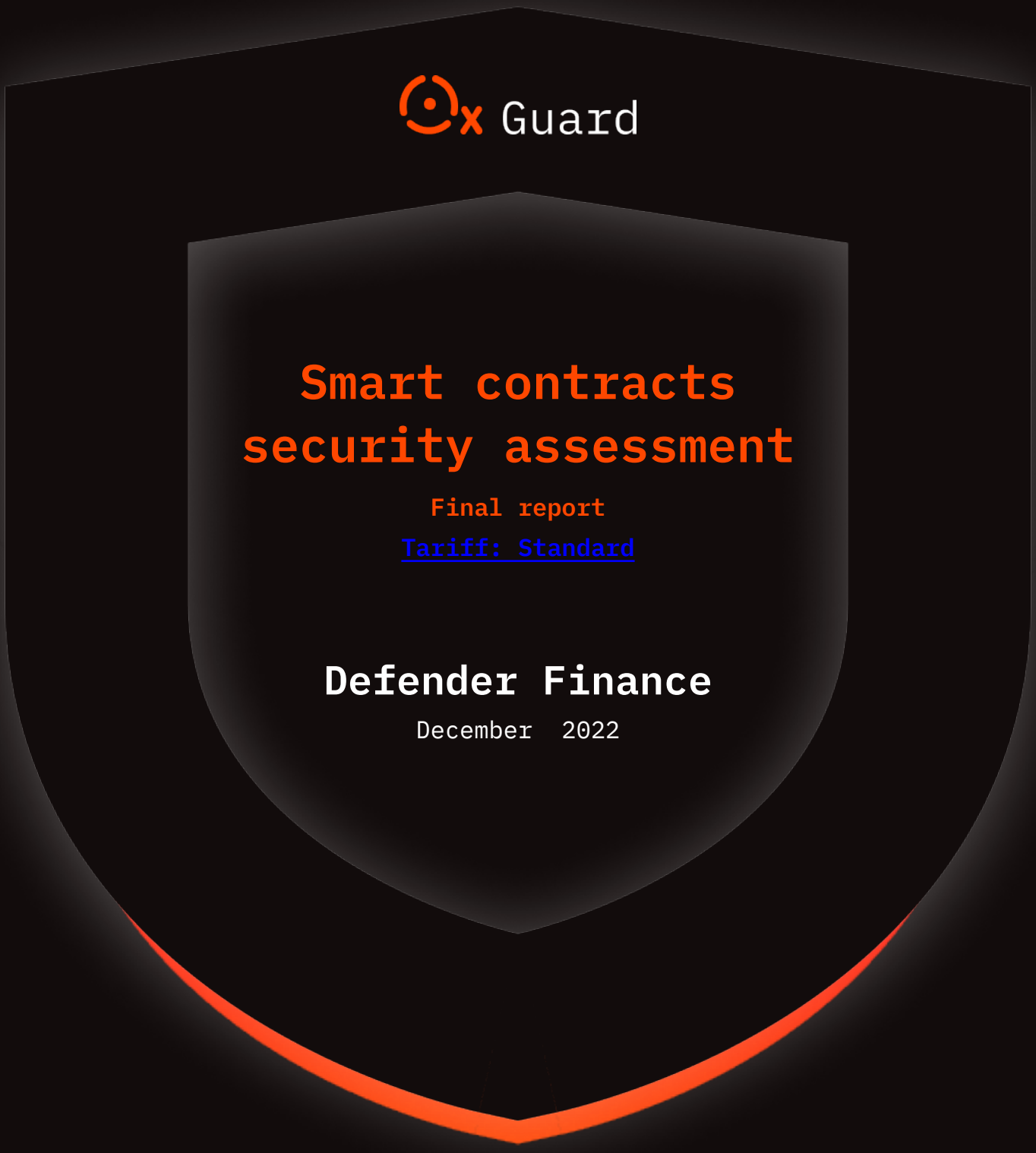

Smart contracts security assessment

Final report

Tariff: Standard

Defender Finance

December 2022

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Defender Finance**.

The Defender Finance Protocol allows users to farm ShareTokens and MainTokens. The MainToken and ShareToken are ERC20-like tokens with taxes on selling. The Maintoken also has a cooldown period (up to 5 minutes) and a maximum amount limit per account on selling.

The ShareTokenRewardPool contracts may charge a fee of up to 1% for each deposit.

Contracts Treasury and Boardroom allow decreasing the price of the MainToken by minting new tokens.

The code is available at the GitHub repository and was audited after the commit 0ad7d697e30ea16daeed3e7fb77acc97744bf05f.

The inspected contracts are: MainToken.sol, ShareToken.sol, ShareTokenRewardPool.sol, Boardroom.sol, Treasury.sol, LPNode.sol.

The ShareTokenRewardPoolV2 contract allows users to farm shareTokens at a different speed for 18 months. The ShareTokenRewardPoolV2 contract may charge a fee of up to 1% for each deposit.

The code of the ShareTokenRewardPoolV2 contract was audited after the commit 63b83d0a992e815ee0f5ab2f0b07f2c0b698969c.

Report Update.

The contract's code was updated according to this report and rechecked after the commit e6b5ad56d37cc019a7da76ded5e069c9a2c94579. The code of the ShareTokenRewardPoolV2 contract was audited after the commit 3c8129d36f20d25cf9a1b60f4f265284dc5ac526.

Name	Defender Finance
Audit date	2022-11-29 - 2022-12-05

Language	Solidity
Platform	Binance Smart Chain

Contracts checked

Name	Address
MainToken	
ShareToken	
ShareTokenRewardPool	
Boardroom	
Treasury	
LPNode	
ShareTokenRewardPoolV2	

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) of 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. Withdrawing tokens by the operator (LPNode)
Status: Fixed

The contract operator can execute the `emergencyWithdraw()` function to withdraw all deposited `lpTokens`.

Recommendation: It is necessary to restrict the ability of the operator to withdraw `lpToken` using `emergencyWithdraw()`.

2. Blocking when calculating rewards (ShareTokenRewardPoolV2)
Status: Fixed

1. Rewards are calculated and updated by iterating over all reward periods (months). To do this, the period is first calculated, and then iteration is carried out over it.

When iterating over the last periods, the array goes beyond the boundaries, which will fail and block the execution of the function.

This behavior is possible due to two factors. First, while iterating, the loop may go beyond the array due to the '`i <= toMonth`' condition. Secondly, at each iteration step, the next element of the array is read `rewardInfos[i + 1].startTime`.

```
    function getDaoReward(uint256 _fromTime, uint256 _toTime) internal view returns
(uint256) {
        uint256 fromMonth = getMonthFrom(_fromTime);
        uint256 toMonth = getMonthFrom(_toTime);
        uint256 reward = 0;
        for (uint256 i = fromMonth; i <= toMonth; ++i) {
            uint256 timeFrom = _fromTime;
            uint256 timeTo = rewardInfos[i + 1].startTime > _toTime ? _toTime :
rewardInfos[i + 1].startTime;
            reward = reward +
timeTo.sub(timeFrom).mul(rewardInfos[i].rewardPerSecondForDao);
            _fromTime = timeTo;
        }

        return reward;
    }
```

The ussie occurs in functions `getDaoReward()`, `getDevReward()`, `getUserReward()`, `updatePool()`.

2. Also, the `getMonth()` function can return a value of 19 or more for the last period (if the user claims a reward after the end of the reward period).

```
    function getMonth() public view returns (uint256) {
        if (block.timestamp < poolStartTime) return 0;
        return (block.timestamp - poolStartTime) / MONTH;
    }
```

In this case, a similar problem will happen in the functions `deposit()` (L401-L404) and `withdraw()` (L446-L449).

```
    function deposit(uint256 _pid, uint256 _amount) external {
        ...
        uint256 fromMonth = getMonthFrom(lastRewardTime);
        uint256 toMonth = getMonth();
        for (uint256 i = fromMonth; i <= toMonth; ++i) {
            user.rewardDebt[i] =
user.amount.mul(pool.accRewardTokenPerShare[i]).div(1e18)
        }
        ...
    }
```

Recommendation: It is necessary to fix iteration over arrays.

Take into account that the total number of periods is 18, but the iteration starts from zero index. So the last element of the array has index 17.

Also, consider adding an `emergencyWithdraw()` function to withdraw the user's tokens in case of failure.

Medium severity issues

1. Tax bypass (MainToken)

Status: Fixed

The `transferFrom()` function allows charging taxes when someone sells the token.

This condition can be circumvented if you interact directly with the LP-Pair without using a router. Because in this case, users will use the `transfer()` function.

Recommendation: Make sure the code works as intended. Otherwise, you should implement similar functionality for the `transfer()` function.

2. Blocked tranfers (MainToken)

Status: Fixed

The `_transfer()` function updates the mapping `_lastTimeReceiveToken` for a **recipient** on every transfer. Mapping `_lastTimeReceiveToken` can be checked in the function `transferFrom()`. Moreover, the check is now carried out for the **sender** of the tokens.

Thus, it is possible to spam a certain user so that he cannot sell his tokens using the `transferFrom()` function. This can be done by sending him one token every 30 minutes.

```
    function transferFrom(address from, address to, uint256 amount) public virtual
override returns (bool) {
        require(polWallet != address(0),"require to set polWallet address");

        address spender = _msgSender();
        _spendAllowance(from, spender, amount);

        // Selling token
□if(marketLpPairs[to] && !excludedTaxAddresses[from]) {
□□require(excludedAccountSellingLimitTime[from] || block.timestamp >
_lastTimeReceiveToken[from].add(timeLimitSelling), "Selling limit time"); /
□□...
□}
        _transfer(from, to, amount);
        return true;
    }


    function _transfer(address from, address to, uint256 amount) internal virtual
override {
        ...
        _lastTimeReceiveToken[to] = block.timestamp;
        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
```

```
    }
```

Recommendation: We recommend reviewing the architecture for blocking users when selling tokens. Perhaps the `_transfer()` function should track token **senders** instead of **recipients**.

3. Blocked tranfers (ShareToken)

Status: Fixed

The `_transfer()` function updates the mapping `_lastTimeReceiveToken` for a **recipient** on every transfer. Mapping `_lastTimeReceiveToken` can be checked in the function `transferFrom()`. Moreover, the check is now carried out for the **sender** of the tokens.

Thus, it is possible to spam a certain user so that he cannot sell his tokens using the `transferFrom()` function. This can be done by sending him one token every 30 minutes.

Recommendation: We recommend reviewing the architecture for blocking users when selling tokens. Perhaps the `_transfer()` function should track token **senders** instead of **recipients**.

4. Tax bypass (ShareToken)

Status: Fixed

The `transferFrom()` function allows charging taxes when someone sells the token.

But this condition can be circumvented if you interact directly with the LP-Pair without using a router. Because in this case, users will use the `transfer()` function.

Recommendation: Make sure the code works as intended. Otherwise, you should implement similar functionality for the `transfer()` function.

Low severity issues

1. Gas optimization (MainToken)
Status: Fixed

1. The `taxOffice` variable can be declared as `immutable` to save gas.

2. The functionality of the `_isExcluded` mapping is never used in the project. Also the defined events `DisableRebase` and `EnableRebase` are never used either. Removing mentioned code will allow to save gas.

2. Gas optimization (ShareToken)
Status: Fixed

The functionality of the `_isExcluded` mapping is never used in the project. Also the defined events `DisableRebase` and `EnableRebase` are never used either. Removing mentioned code will allow to save gas.

3. Source of rewards (ShareTokenRewardPool)
Status: Fixed

The contract rewards were defined on L48-50. At the same time, the source of such awards is not explicitly indicated.

Recommendation: We recommend double-checking reward sizes and sources and adding documentation to them.

4. Variable default visibility (ShareTokenRewardPool)
Status: Fixed

The variables `lastDaoFundRewardTime`, `lastDevFundRewardTime` have default visibility. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

5. Gas optimization (ShareTokenRewardPool)
Status: Fixed

1. The variables `devFund`, `daoFund`, `polWallet` can be declared as `immutable` to save gas.

2. The imported module `ContractGuard` (L5) is never used in the contract code and can be removed.

3. The structure `DevFundInfo` is never used in the contract code and can be removed.

6. Unused functionality (Boardroom)
Status: Fixed

The variable `rewardDebt` of the `Memberseat` structure is never used and can be removed.

7. Validation in the initialize() function (Treasury)
Status: Fixed

1. We recommend adding validation `pairAddress != address(0)` on L203 to prevent incorrect pair initialization.

2. Consider retrieving the `STABLE_DECIMALS` value directly from the stable token via an external call (for example: `IERC20Metadata(_stableToken).decimals()`).

8. Allowance update (Treasury)
Status: Fixed

The `_approveTokenIfNeeded()` function allows updating the allowance if it is 0.

But a situation may occur when the allowance does not equal zero and is too small for the transaction.

To do this, we recommend adding a second parameter to the function - the required amount of `allowance`.

9. Gas optimization (Treasury)
Status: Fixed

The variable ONE (L63) is never used in the contract code and can be removed.

10. Constructor lacks validation of input parameters (LPNode)
Status: Open

The contract constructor does not check the `_startTime` and `_medalRate` values.

11. Updating startTime (LPNode)

Status: Fixed

The `setStartTime()` function allows to update the start time of the contract.

Unlike the contract constructor, this function does not update the `lastDeliveryTime` variable. Such implementation may lead to the incorrect distribution of rewards.

Recommendation: Consider updating the `lastDeliveryTime` variable in the `setStartTime()` function.

12. Gas optimization (LPNode)

Status: Fixed

1. The variable `lpToken` can be declared as `immutable` to save gas.

2. The visibility of the `setStartTime()`, `totalUsers()` functions can be changed to `external` type to save gas.

13. Gas optimization (ShareTokenRewardPoolV2)

Status: Fixed

1. Variables `poolStartTime` and `poolEndTime` can be declared as `immutable` to save gas.

2. The `getUserReward()` function calculates rewards for all specified periods. The local variable `tokenSupply` is used to calculate accumulated rewards for each period. Since the variable `tokenSupply` does not change during the execution of the function, it is enough to define it only once, outside the loop. We recommend moving the L301 out from the for-loop.

3. The visibility of the functions `add()`, `set()` and `setDepositFeePercent()` can be declared `external` instead of `public` to save gas.

Conclusion

Defender Finance MainToken, ShareToken, ShareTokenRewardPool, Boardroom, Treasury, LPNode, ShareTokenRewardPoolV2 contracts were audited. 2 high, 4 medium, 13 low severity issues were found.
2 high, 4 medium, 12 low severity issues have been fixed in the update.

The contract owner (operator) can set a fee of up to 0.5% for selling tokens.

We recommend writing tests to cover the founded issues and recheck pools' rewards.

Also, consider adding a view function for MainToken to track the current `maxAmountSell` value.

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.
OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

Slither output

```
ShareTokenRewardPool.pending(uint256,address) (contracts/
ShareTokenRewardPool.sol#178-194) performs a multiplication on the result of a
division:
        - _shareTokenReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint) (contracts/
ShareTokenRewardPool.sol#185)
        - accShareTokenPerShare =
accShareTokenPerShare.add(_shareTokenReward.mul(1e18).div(tokenSupply)) (contracts/
ShareTokenRewardPool.sol#186)
ShareTokenRewardPool.updatePool(uint256) (contracts/ShareTokenRewardPool.sol#239-259)
performs a multiplication on the result of a division:
        - _shareTokenReward =
_generatedReward.mul(pool.allocPoint).div(totalAllocPoint) (contracts/
ShareTokenRewardPool.sol#255)
        - pool.accShareTokenPerShare =
pool.accShareTokenPerShare.add(_shareTokenReward.mul(1e18).div(tokenSupply)) (contracts/
ShareTokenRewardPool.sol#256)
Treasury.getEstimatedReward(uint256) (contracts/Treasury.sol#298-321) performs a
multiplication on the result of a division:
        - mainTokenReward = mainTokenCirculatingSupply.mul(percentage).div(100).div(10
** STABLE_DECIMALS) (contracts/Treasury.sol#301)
        - mainTokenReward = mainTokenReward.mul(expansionRate).div(10000) (contracts/
Treasury.sol#303)
Treasury.getEstimatedReward(uint256) (contracts/Treasury.sol#298-321) performs a
multiplication on the result of a division:
        - mainTokenReward = mainTokenReward.mul(expansionRate).div(10000) (contracts/
Treasury.sol#303)
        - mainTokenAmountToSell =
mainTokenReward.mul(mainTokenPercentToSell).div(10000) (contracts/Treasury.sol#305)
Treasury.getEstimatedReward(uint256) (contracts/Treasury.sol#298-321) performs a
multiplication on the result of a division:
        - medalEstimated = medalEstimated.mul(boardroomRewardPercent).div(10000)
(contracts/Treasury.sol#315)
        - medalEstimated.mul(boardroomPool.allocPoint).div(totalAllocPoint) (contracts/
Treasury.sol#316)
Treasury.allocateSeigniorage() (contracts/Treasury.sol#442-468) performs a
multiplication on the result of a division:
```

```
        - totalTokenExpansion =
mainTokenCirculatingSupply.mul(percentage).div(100).div(10 ** STABLE_DECIMALS)
(contracts/Treasury.sol#451)
        - totalTokenExpansion = totalTokenExpansion.mul(expansionRate).div(10000)
(contracts/Treasury.sol#455)
LPNode.getDayRewardEstimate(address) (contracts/lpnode/LPNode.sol#277-294) performs a
multiplication on the result of a division:
        - rewardPerSecond =
getBalancePool().mul(rewardPercent).div(PERCENTAGE).div(rollingDuration) (contracts/
lpnode/LPNode.sol#284-287)
        - dayRewardEstimate =
rewardPerSecond.mul(86400).mul(userAllocPoints).div(totalAllocPoints()) (contracts/
lpnode/LPNode.sol#289-292)
LPNode._burnTokenByRate(address,uint256) (contracts/lpnode/LPNode.sol#367-374) performs
a multiplication on the result of a division:
        - medalAmount =
((_amount.mul(MULTIPLIER).div(medalRate)).mul(PERCENTAGE)).div(maxReturnPercent)
(contracts/lpnode/LPNode.sol#371-372)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-
multiply

ShareTokenRewardPool.updatePool(uint256) (contracts/ShareTokenRewardPool.sol#239-259)
uses a dangerous strict equality:
        - tokenSupply == 0 (contracts/ShareTokenRewardPool.sol#245)
Treasury._sellMainTokenAndAddLp() (contracts/Treasury.sol#323-376) uses a dangerous
strict equality:
        - mainTokenBalanceOf == 0 (contracts/Treasury.sol#326)
LPNode._compound(address) (contracts/lpnode/LPNode.sol#441-479) uses a dangerous strict
equality:
        - _rewards == 0 (contracts/lpnode/LPNode.sol#453)
LPNode.getPendingRewards(address) (contracts/lpnode/LPNode.sol#244-266) uses a
dangerous strict equality:
        - users[_sender].totalDeposits == 0 (contracts/lpnode/LPNode.sol#249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-
strict-equalities

Reentrancy in LPNode.claimPolRewards() (contracts/lpnode/LPNode.sol#172-178):
        External calls:
        - lpToken.safeTransfer(pol,polRewards) (contracts/lpnode/LPNode.sol#174)
        State variables written after the call(s):
        - polRewards = 0 (contracts/lpnode/LPNode.sol#175)
```

```
Reentrancy in Treasury.initialize(address,address,address,address,address,address,addres
s,address,uint256,uint256) (contracts/Treasury.sol#176-224):
        External calls:
        - IMainToken(mainToken).grantRebaseExclusion(address(this)) (contracts/
Treasury.sol#219)
        State variables written after the call(s):
        - initialized = true (contracts/Treasury.sol#221)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-1

Treasury.calculateExpansionRate(uint256) (contracts/Treasury.sol#426-440) contains a
tautology or contradiction:
        - tierId >= 0 (contracts/Treasury.sol#432)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-
contradiction

Treasury.calculateExpansionRate(uint256).expansionRate (contracts/Treasury.sol#427) is
a local variable never initialized
Treasury.getMainTokenPrice().price (contracts/Treasury.sol#169) is a local variable
never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-
local-variables

Treasury.getMainTokenPrice() (contracts/Treasury.sol#168-174) ignores return value by
IOracle(oracle).consult(mainToken,1e18) (contracts/Treasury.sol#169-173)
Treasury._sellMainTokenAndAddLp() (contracts/Treasury.sol#323-376) ignores return value
by ROUTER.swapExactTokensForTokens(mainTokenAmountToSell,0,path,address(this),block.time
stamp) (contracts/Treasury.sol#334-340)
Treasury._sellMainTokenAndAddLp() (contracts/Treasury.sol#323-376) ignores return value
by ROUTER.addLiquidity(mainToken,stableToken,mainTokenBalanceOf.sub(mainTokenAmountToSel
l),stableTokenBalanceOf,0,0,address(this),block.timestamp) (contracts/
Treasury.sol#346-355)
Treasury._depositToLPNode() (contracts/Treasury.sol#378-384) ignores return value by
PAIR.approve(medalPool,pairAmount) (contracts/Treasury.sol#381)
Treasury.allocateSeigniorage() (contracts/Treasury.sol#442-468) ignores return value by
IMainToken(mainToken).mint(address(this),totalTokenExpansion) (contracts/
Treasury.sol#457)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

MainToken.revokeRebaseExclusion(address) (contracts/MainToken.sol#93-106) has costly
operations inside a loop:
```

```
        - excluded.pop() (contracts/MainToken.sol#101)
ShareToken.revokeRebaseExclusion(address) (contracts/ShareToken.sol#86-99) has costly
operations inside a loop:
        - excluded.pop() (contracts/ShareToken.sol#94)
ShareTokenRewardPool.updatePool(uint256) (contracts/ShareTokenRewardPool.sol#239-259)
has costly operations inside a loop:
        - totalAllocPoint = totalAllocPoint.add(pool.allocPoint) (contracts/
ShareTokenRewardPool.sol#251)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-
operations-inside-a-loop

Pragma version0.8.13 (contracts/Boardroom.sol#3) allows old versions
Pragma version0.8.13 (contracts/MainToken.sol#3) allows old versions
Pragma version0.8.13 (contracts/ShareToken.sol#3) allows old versions
Pragma version0.8.13 (contracts/ShareTokenRewardPool.sol#3) allows old versions
Pragma version0.8.13 (contracts/Treasury.sol#3) allows old versions
Pragma version^0.8.0 (contracts/lpnode/LPNode.sol#2) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity

Treasury.ONE (contracts/Treasury.sol#63) is never used in Treasury (contracts/
Treasury.sol#20-557)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-
variable
```

0x Guard