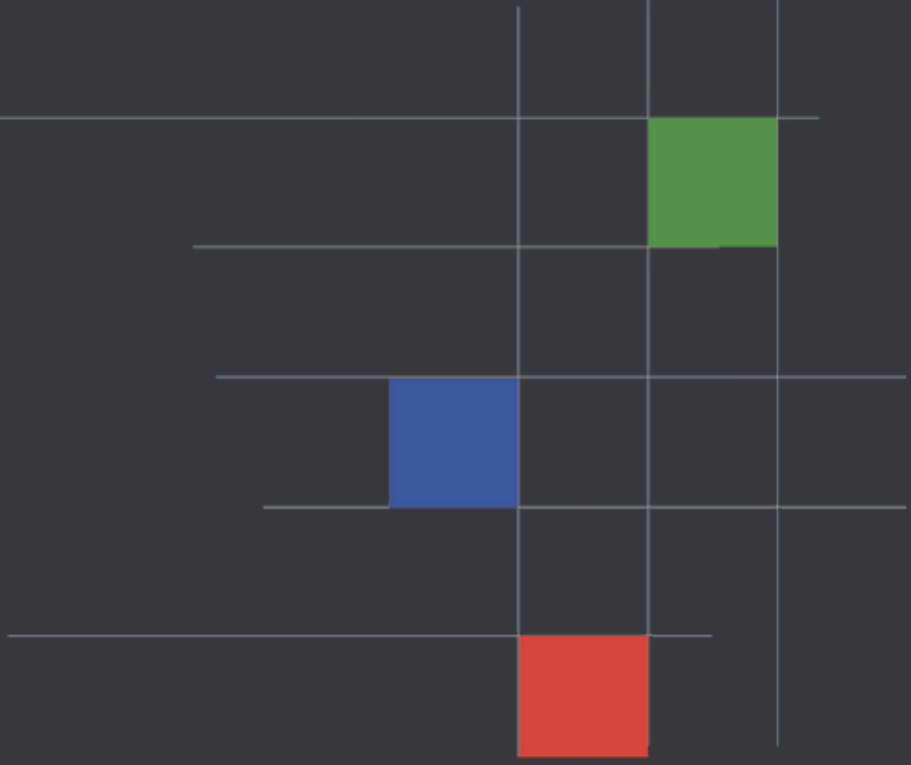
bDollar Protocol

Security Assessment

February 24th, 2021

For :
bDollar Protocol



Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 CertiK to perform a security review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	bDollar
Description	an algorithmic stablecoin
Platform	Binance Smart Chain; Solidity; Yul
Codebase	GitHub Repository
Commits	1ea9f3ad2036a13b3ec7dcb4c98d9770b0bf18fc

Audit Summary

Delivery Date	Feb. 24th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 4, 2021 - Feb. 13, Feb. 24, 2021

Vulnerability Summary

Total Issues	13
Total Critical	0
Total Major	0
Total Minor	5
Total Informational	7
Total Discussion	1

Executive Summary

This report has been prepared for **bDollar** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

All of the functions in the protocol have proper access restriction and parameter sanitization where necessary. The equity was found to be calculated correctly for each of the accounts. Most of the findings are optimizational.

Additionally, to bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following capability of the `operator`:

- `operator` can transfer assests from `CommunityFund` to any addresses.
- `operator` can call any internal functions via `executeTransaction()` method in `CommunityFund` contract.
- The deployment address of `CommunityFund` contract is an upgradeable proxy, and the `operator` is set to the `Bearn.fi: Deployer`. The `operator` address is defined within the initialization process. Hence each time this contract is upgraded for any governace proposal, it will execute initialization process again.

To improve the trustworthiness of the project, any dynamic runtime changes on the protocol should be notified to clients. Any plan to call these methods is better to move to the execution queue of Timelock, and also emit events.

The bDollar team confirmed that `operator` role will be transfered to Multisig soon when `BDOIP02` completed and later will completely transfered to DAO.

File in Scope

ID	Contract	SHA-256 Checksum
DOL	Dollar.sol	e8a8385695679e33059abdd46ca14820a07ca4b1d14c725a3640b40244cb2852
SHA	Share.sol	f3e0a62376f522dbc0e01a4715f03b1d92bfe13f513dfb9ca6d8476cd6aa3350
BOD	Bond.sol	b6225f43b8e3f4f65399fa82287e209ef4fda496c250dc1daaf02020fd144177
BRP	distribution/BdoRewardPool.sol	6b0b98a4bd00c1e106a2aabab40fde562bf09d9d07814354d0c1390a088be048
SRP	distribution/ShareRewardPool.sol	fd156d6a0147e4e869a43510cf363337446c4a904d35443366a552aa35272ca6
OSP	OracleSinglePair.sol	23a68a53e7733aec59e4ce8089272c1699e8374bd6af9d020566243ee4bc5232
OMP	OracleMultiPair.sol	0c9ef16b0a75736977973c4b9f8beffd5b776d1f3357de77a3eb414e23a29985
BRM	Boardroom.sol	84f5593e0be0aea97d5b6fc3b5f5040b4fbfbb80ebf269ae14fb38b5bab5d22a
TSY	Treasury.sol	1590650e9ce2f03be107300b63426fb88810721027787ff4dfa1f1dddcbbd554
CFD	dao/CommunityFund.sol	b07170fcfb4fe41b63f5d2bff48b6d467b45a34b13ef0f810a78175914840fcc

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **bDollar** team or reported an issue.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 5 minor vulnerabilities were identified during our audit that solely concerns the specification.

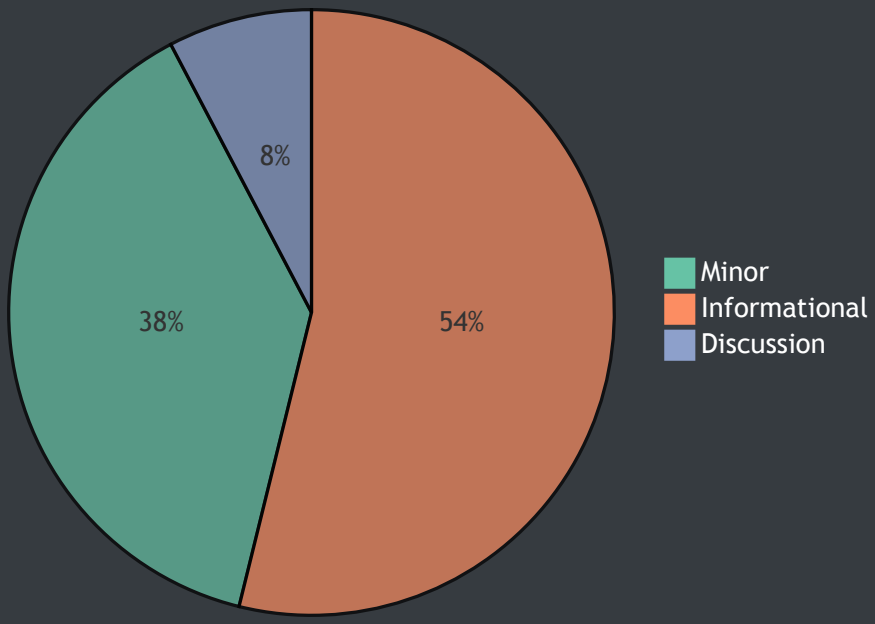
Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate documentation and specification outside of the source files, also the code comment coverage is detailed.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings



ID	Title	Type	Severity
BRP-01	Transfer Balance less than `_amount`	Logical Issue	Minor
BRP-02	Missing Emit Events	Optimization	Minor
BRP-03	Missing Some Important Checks	Logical Issue	Minor
BRP-04	Simplifying Existing Code	Optimization	Informational
BRP-05	Logical Issue in Function `add`	Logical Issue	Minor
BRP-06	Logical Issue in Function `getGeneratedReward`	Logical Issue	Minor
DOL-01	Proper Usage of “public” and “external” Type	Gas Optimization	Informational
SHA-01	State Variables that could be Declared Constants	Gas Optimization	Informational
SHA-02	Code Redundancy	Dead Code	Informational
TSY-01	State Variable never Initialized before Usage	Dead Code	Informational
TSY-02	Incorrect Naming Convention Utilization	Coding Style	Informational
CFD-01	Governance DAO Capability	Optimization	Discussion
OMP-01	Inappropriate Usage of Uniswap	Volatile Code	Informational

BRP-01: Transfer Balance less than `_amount`

Type	Severity	Location
Logical Issue	Minor	BdoRewardPool.sol L261

Description:

```
        function safeBdoTransfer(address _to, uint256 _amount) internal {
            uint256 _bdoBal = bdo.balanceOf(address(this));
            if (_bdoBal > 0) {
                if (_amount > _bdoBal) {
                    bdo.safeTransfer(_to, _bdoBal);
                } else {
                    bdo.safeTransfer(_to, _amount);
                }
            }
        }
```

According to the `safeBdoTransfer()` , its intention is to transfer the `_amount` from `BdoRewardPool` contract to the desired `_to` address.
If the `BdoRewardPool` contract has enough balance, it will transfer `_amount` to the destination.
Else, it will just transfer the `_bdoBal` to the destination.

Recommendation:

The advantage of `safeBdoTransfer()` method in the protocol is that the administrator reserves the ability to transfer the main part of the amount to desitination under unexpected cases. It is also worthy of note the downside of `safeBdoTransfer()` method, where the destination address migth not receive the whole amount.

Should emit events with parameters `_to` and `_amount` in case `_amount > _bdoBal` .

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

BRP-02: Missing Emit Events

Type	Severity	Location
Optimization	Minor	BdoRewardPool.sol, ShareRewardPool.sol, Boardroom.sol, Treasury.sol, CommunityFund.sol

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `constructor()` , `setOperator()` , `add()` , `set()` in `BdoRewardPool` and `ShareRewardPool` contracts;
`setOperator()` , `setBoardroom()` , `setDollarOracle()` , `setDollarPriceCeiling()` , `setBondDepletionFloorPercent()` , `setMaxSupplyContractionPercent()` , `setMaxDeptRatioPercent()` ...etc in `Treasury` contract; `setOperator()` , `setLockUp()` in `Boardroom` contract; `setOperator()` , `setStrategist()` , `setTreasury()` , `setShareRewardPool()` ...etc in `CommunityFund` contract.

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
        event SetOperator(address indexed _operator);
        function setOperator(address _operator) external onlyOperator {
            operator = _operator;
            emit SetOperator (_operator);
        }
```

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

BRP-03: Missing Some Important Checks

Type	Severity	Location
Logical Issue	Minor	BdoRewardPool.sol, ShareRewardPool.sol, CommunityFund.sol, Treasury.sol, Boardroom.sol

Description:

Functions like `set()` , `pendingBDO()` , `updatePool()` , `pendingShare()` ...etc in contract `BdoRewardPool` and `ShareRewardPool` are missing parameter `_pid` checks.

```
      PoolInfo storage pool = poolInfo[_pid];
```

If `_pid` is invalid, the above sentence will throw an error.

In contract `CommunityFund` , function `claimRewardFromSharePool()` behaves like an internal function. If it is public function, it is missing parameter address zero check. If it is private function, better to change the function as private.

Function `setOperator()` in `CommunityFund` , `Treasury` , `Boardroom` , `BdoRewardPool` and `ShareRewardPool` contracts are missing parameter address zero check.
Function `setTreasuryFund()` in `Share` contracts is missing parameter address zero check.
Function `initialize()` in `Boardroom` contracts is missing parameter address zero check.

Recommendation:

We recommend to add neccessary checks, for example:

```
        function set(uint256 _pid, uint256 _allocPoint) public onlyOperator {
            ...
            require(_pid<poolInfo.length, “BdoRewardPool: pool doesn’t exist!”);
            PoolInfo storage pool = poolInfo[_pid];
            ...
        }
        function setOperator(address _operator) external onlyOperator {
            require(_operator != address(0x0), “CommunityFund: _operator cannot be address
    zero!”);
            operator = _operator;
        }
```

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

BRP-04: Simplifying Existing Code

Type	Severity	Location
Optimization	Informational	BdoRewardPool.sol L104-L118, ShareRewardPool.sol L97-L111

Description:

Consider using a more efficient coding snipit to replace the below codes:

```
        if (block.number < startBlock) {
            // chef is sleeping
            if (_lastRewardBlock == 0) {
                _lastRewardBlock = startBlock;
            } else {
                if (_lastRewardBlock < startBlock) {
                    _lastRewardBlock = startBlock;
                }
            }
        } else {
            // chef is cooking
            if (_lastRewardBlock == 0 || _lastRewardBlock < block.number) {
                _lastRewardBlock = block.number;
            }
        }
```

Recommendation:

Consider changing it as following example:

```
        _lastRewardBlock = block.number > startBlock ? block.number : startBlock;
```

Alleviation:

The recommendation was not taken into account, with the bDollar team stating it could not be replaced by the recommendation code, because they need ability to set `_lastRewardBlock` to a future block (not `startBlock` nor current `block.number`).

BRP-05: Logical Issue in Function `add`

Type	Severity	Location
Logical Issue	Minor	BdoRewardPool.sol L119, ShareRewardPool.sol L112

Description:

Below codes will always initialize `_isStarted` as `true`.

```
        bool _isStarted =
           (_lastRewardBlock <= startBlock) ||
           (_lastRewardBlock <= block.number);
```

Recommendation:

Consider changing it as following example:

```
       bool _isStarted = block.number >= startBlock;
```

Alleviation:

The recommendation was not taken into account, with the bDollar team stating if `_lastRewardBlock` is a future block then it does not work with the recomendation code.

BRP-06: Logical Issue in Function `getGeneratedReward`

Type	Severity	Location
Logical Issue	Minor	BdoRewardPool.sol L147, ShareRewardPool.sol L140

Description:

The function `getGeneratedReward()` doesn't check whether parameter `_from` is greater than `startBlock` or not. And it is a public function, in case the `_from` is smaller than `startBlock`, the function will return wrong result.

```
    function getGeneratedReward(uint256 _from, uint256 _to) public view returns (uint256) {
        ...
        return _to.sub(_from).mul(epochBdoPerBlock[0]);
    }
```

Recommendation:

Consider adding logic to check parameter `_from` comparing with `startBlock`, or changing it to private function.

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

DOL-01: Proper Usage of "public" and "external" Type

Type	Severity	Location
Gas Optimization	Informational	Dollar.sol, BdoRewardPool.sol, Boardroom.sol, Treasury.sol

Description:

"public" functions that are never called by the contract could be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.
Examples:
Function `mint()` in contract `Dollar` .
Functions `set()` , `deposit()` , `withdraw()` , `emergencyWithdraw()` in contract `BdoRewardPool` .
Functions `initialize()` , `rewardPerShare()` in contract `Boardroom` .
Functions `isMigrated()` , `isInitialized()` , `getDollarUpdatedPrice()` , `getReserve()` , `getBurnableDollarLeft()` , `getRedeemableBonds()` , `initialize()` in contract `Treasury` .

Recommendation:

Consider using the "external" attribute for functions never called from the contract.

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

SHA-01: State Variables that could be Declared Constants

Type	Severity	Location
Gas Optimization	Informational	Share.sol L19,L22,L23

Description:

Constant state variables should be declared constant to save gas.

```
    uint256 public startTime = 1608811200;
    uint256 public communityFundRewardRate = COMMUNITY_FUND_POOL_ALLOCATION /
VESTING_DURATION;
    uint256 public devFundRewardRate = DEV_FUND_POOL_ALLOCATION / VESTING_DURATION;
    uint256 public runningBlocks = 10625000; // 368 days
    uint256 public sbdoPerBlock = 0.008 ether;
```

Recommendation:

Add constant attributes to state variables that never change.

We recommend to change the codes like below examples:

```
    uint256 public constant COMMUNITY_FUND_REWARD_RATE = COMMUNITY_FUND_POOL_ALLOCATION /
VESTING_DURATION;
```

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

SHA-02: Code Redundancy

Type	Severity	Location
Dead Code	Informational	Share.sol L49-L61

Description:

Below 2 functions can be combined into 1 function with parameter.

```
        function unclaimedTreasuryFund() public view returns (uint256 _pending) {
            uint256 _now = block.timestamp;
            if (_now > endTime) _now = endTime;
            if (communityFundLastClaimed >= _now) return 0;
            _pending = _now.sub(communityFundLastClaimed).mul(communityFundRewardRate);
        }

        function unclaimedDevFund() public view returns (uint256 _pending) {
            uint256 _now = block.timestamp;
            if (_now > endTime) _now = endTime;
            if (devFundLastClaimed >= _now) return 0;
            _pending = _now.sub(devFundLastClaimed).mul(devFundRewardRate);
        }
```

Recommendation:

We recommend to change the code as below:

```
        function unclaimed(uint256 _fundLastClaimed, uint256 _fundRewardRate) public view
    returns (uint256 _pending) {
            uint256 _now = block.timestamp;
            if (_now > endTime) _now = endTime;
            if (_fundLastClaimed >= _now) return 0;
            _pending = _now.sub(_fundLastClaimed).mul(_fundRewardRate);
        }
```

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

TSY-01: State Variable never Initialized before Usage

Type	Severity	Location
Dead Code	Informational	Treasury.sol L461

Description:

Some variables do not initialize before usage.

```
        uint256 _savedForBond;
```

Recommendation:

Consider initializing all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero.

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

TSY-02: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	Treasury.sol L65, Share.sol L38

Description:

According to the logic, state variable `maxDeptRatioPercent` in contract `Treasury` is better to be named as `maxDebtRatioPercent`.

Functions `setTreasuryFund()`, `unclaimedTreasuryFund()` in contract `Share` are better to be named as `setCommunityFund()`, `unclaimedCommunityFund()`.

Solidity defines a naming convention that should be followed. Refer to: https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions

Recommendation:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

CFD-01: Governance DAO Capability

Type	Severity	Location
Optimization	Discussion	CommunityFund.sol L163,L424

Description:

Function `grandFund()` in contract `CommunityFund` is only callable by `operator`. This `grandFund()` can transfer assests from `CommunityFund` to any addresses.
Secondly, operator can call any internal functions via `executeTransaction()` method in this contract.
The deployment address of `CommunityFund` contract is an upgradeable proxy, and the `operator` is set to the `Bearn.fi: Deployer`. The `operator` address is defined within the initialization process. Hence each time this contract is upgraded for any governace proposal, it will execute initialization process again.

Additionally, to bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness.

To improve the trustworthiness of the project, any dynamic runtime changes on the protocol should be notified to clients. Any plan to call these methods is better to move to the execution queue of Timelock, and also emit events.

Is there any plan to move the `operator` role to Timelock contract? Or make the `operator` Multi-signers just as the governace proposal `BDOIP02` ?

Alleviation:

The team confirmed that operator role will be transfered to Multisig soon when BDOIP02 completed and later will completely transfered to DAO.

OMP-01: Inappropriate Usage of Uniswap

Type	Severity	Location
Logic Issue	Informational	OracleMultiPair.sol L134

Description:

Function `getOtherTokenAmountFromPair()` relies on the reserves in PancakePair. Incase the pair was attacked by flash loan, the price is accumulative and does not fear the flash loan attack. But the reserves are not accumulative.

Recommendation:

We recommended to do calculations based on the price from PancakePair.

Alleviation:

The team heeded our advice and will fix the issue in their own timeframe.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.