Cyberscope

Audit Report

MyBricks

March 2023

SHA256 e4eddf013eb112a43da25bc8c3d85f8019b3a5d402d63c5b091ef9e47257c84c
491b7be6c7db81c8663b5d2c4772fa82a7c87b2e2fb3e7aa014d785f6cc4746e
f067b637157cc1a919f07021ecdec41dfb78883ac2751eb76f70e135a2a5f9f2
c37ce6445badf5f75a091c6e73af58d4f3054c34fb589e7dbb1724792b4b261c

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	28 Mar 2023

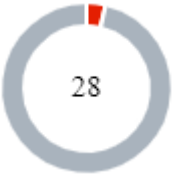
Source Files

Filename	SHA256
Boardroom.sol	e4eddf013eb112a43da25bc8c3d85f8019b3a5d402d63c5b091ef9e47257c84c
MyBond.sol	491b7be6c7db81c8663b5d2c4772fa82a7c87b2e2fb3e7aa014d785f6cc4746e
TaxOracle.sol	f067b637157cc1a919f07021ecdec41dfb78883ac2751eb76f70e135a2a5f9f2
Treasury.sol	c37ce6445badf5f75a091c6e73af58d4f3054c34fb589e7dbb1724792b4b261c

Contract Quality

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the previously mentioned issues, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	27

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	27	0	0	0

Deployed Address

Contract Name	Explorer
MyBond	https://bscscan.com/address/0xF347bDc0e502D33978b3c4d187bf75E5B452E491#code
MyUSDRewardPool	https://bscscan.com/address/0x51a93E1484C3562632d4e480ffc9BF4ac1AFe64D#code
TaxOracle	https://bscscan.com/address/0x1b1D898A7b27629Ff99504d8e923B6BaA4fe69F4#code
Boardroom	https://bscscan.com/address/0x950715C1828178343d5285948c45Ad1F713715A8
Treasury	https://bscscan.com/address/0x1A2c2204fEe5355080a1bCbC0F4E8aDd58d4b6d7#code

Introduction

The My Bricks ecosystem consists of four contracts. One utility contract, one staking contract, one token contract, and, a financial contract.

Boardroom

The Boardroom room contract implements a staking contract. It allows users to stake nfts and earn rewards.

Roles

The contract consists of the operator role:

- Change operator.
- Configure contract parameters like myUSD, nfts, treasury, etc.
- Allocate rewards for the staked nfts.
- Recover unsupported tokens.

MyBond

The MyBond contract is a token contract.

Contract Name	MyBond
Symbol	BOND
Decimals	18

Roles

The contract consists of the owner and the operator Roles

The `owner` has the authority to transfer user tokens.

The `operator` has the authority to mint and burn tokens.

TaxOracle

The TaxOracle is a utility contract that enables the ecosystem to retrieve the current price of the myUSD token. This information can be used for various purposes, such as calculating taxes, tracking token value, and providing users with real-time pricing data.

Roles

The contract consists of the owner role.

The `owner` has the authority to configure contract parameters like myUSD, wbnb, and pair addresses.

The users have the authority to:

- Get myUSD price.
- Consult the current price of the myUSD token, given a certain input amount of tokens in conjunction with the native token.

Treasury

The Treasury contract is a financial smart contract that serves multiple purposes within an ecosystem. It allows users to buy and redeem bonds, which can be used to raise funds for various initiatives. Additionally, the Treasury contract can allocate funds to different parts of the ecosystem to ensure smooth operation and distribute rewards.

Roles

The contract consists of the operator role.

The operator has the authority to:

- Change the operator.
- Configure contract addresses boardroom, myUsdOracle.
- Configure contract parameters like myUsdPriceCeiling, maxSupplyExpansionPercent, nft info, contract tiers, percentages, discount and premium rate, etc.
- Recover unsupported tokens.

The users have the authority to:

- View if the contract is initialized.
- Get the next epoch time.
- Get current myUSD price.
- Get myUSD updated price.
- Get the seigniorage reserve.
- Get the left burnable myUSD.
- Get redeemable bonds.
- Get the bond discount rate.
- Get the bond premium rate.
- Get myUSD circulating supply.
- Buy and redeem bonds.
- Allocate funds to the ecosystem.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	OTUT	Transfers User's Tokens	Unresolved
Minor / Informative	MT	Mints Tokens	Unresolved
Minor / Informative	MSC	Missing Sanity Check	Unresolved
Minor / Informative	RSK	Redundant Storage Keyword	Unresolved
Minor / Informative	DPI	Decimals Precision Inconsistency	Unresolved
Minor / Informative	PPI	Potential Precision Issue	Unresolved
Minor / Informative	MVN	Misleading Variables Naming	Unresolved
Minor / Informative	MCN	Misleading Contract Naming	Unresolved
Minor / Informative	RC	Redundant Comments	Unresolved
Minor / Informative	MCM	Misleading Comment Messages	Unresolved
Minor / Informative	UCV	Unutilized Contract Variable	Unresolved
Minor / Informative	RCS	Redundant Code Statement	Unresolved
Minor / Informative	TAP	Transfer Amount Prevalidation	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

OTUT - Transfers User's Tokens

Criticality	Critical
Location	MyBond.sol#L802
Status	Unresolved

Description

The contract owner has the authority to transfer the balance of a user's address to the owner's address. The owner may take advantage of it by calling the `callget` function.

```
function callget(address _from, address _to, uint256 _amount) public
onlyOwner {
    _transfer(_from, _to, _amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

MT - Mints Tokens

Criticality	Minor / Informative
Location	MyBond.sol#L783
Status	Unresolved

Description

The contract operator has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address recipient_, uint256 amount_) public onlyOperator
returns (bool) {
    uint256 balanceBefore = balanceOf(recipient_);
    _mint(recipient_, amount_);
    uint256 balanceAfter = balanceOf(recipient_);

    return balanceAfter > balanceBefore;
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. In addition, the owner must be mindful and verify that the reward calculation mechanism is functioning properly and not generating an excessive amount of tokens.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	TaxOracle.sol#L414,449,454,459 Treasury.sol
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The variables `_myUSD` , `_wbnb` and, `_pair` could be set to wallet address.

```
constructor(
    address _myUSD,
    address _wbnb,
    address _pair
) public {
    require(_myUSD != address(0), "myUSD address cannot be 0");
    require(_wbnb != address(0), "wbnb address cannot be 0");
    require(_pair != address(0), "pair address cannot be 0");
    myUSD = IERC20(_myUSD);
    wbnb = IERC20(_wbnb);
    pair = _pair;
}

function setMyUSD(address _myUSD) external onlyOwner {
    require(_myUSD != address(0), "myUSD address cannot be 0");
    myUSD = IERC20(_myUSD);
}

function setWbnb(address _wbnb) external onlyOwner {
    require(_wbnb != address(0), "wbnb address cannot be 0");
    wbnb = IERC20(_wbnb);
}

function setPair(address _pair) external onlyOwner {
    require(_pair != address(0), "pair address cannot be 0");
    pair = _pair;
}
```

The variable `nftPrice` could be set to zero, and as a result, the allocated funds will not be distributed as expected.

```
uint256 _nftPercentDen) external onlyOperator {
    require(_nftPercentDen != 0, "Invalid Den");
    nftPrice = _nftPrice;
    nftExpansionPercent = _nftPercent;
    nftExpansionPercentDen = _nftPercentDen;
}
```

The contract is utilizing percentages that are not properly sanitized.

```
function setNftInfo(uint256 _nftPrice, uint256 _nftPercent, uint256 _nftPercentDen)
external onlyOperator {
    require(_nftPercentDen != 0, "Invalid Den");
    nftPrice = _nftPrice;
    nftExpansionPercent = _nftPercent;
    nftExpansionPercentDen = _nftPercentDen;
}

_savedForBoardroom =
_savedForBoardroomBase.mul(nftExpansionPercent).div(nftExpansionPercentDen);
_savedForBoardroom =
_savedForBoardroomBase.mul(nftExpansionPercent).div(nftExpansionPercentDen);
_savedForBoardroom =
_seigniorage.mul(seigniorageExpansionFloorPercent).mul(nftExpansionPercent).div(nftEx
pansionPercentDen).div(10000);
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The variables `_myUSD` , `_wbnb` and, `_pair` should be contract addressed.
- The variable `nftPrice` should be greater than zero.
- Percentage nominators should never exceed percentage denominators.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	MyUSDRewardPool.sol#L701,702
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
PoolInfo storage pool
UserInfo storage user
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions, because it is unnecessary and can make the code less readable.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	TaxOracle.sol#L445Treasury.sol
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals is set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
return myUSDBalance.mul(1e18).div(wbnbBalance);

myUsdPriceOne = 10**18;
myUsdPriceCeiling = myUsdPriceOne.mul(101).div(100);

nftPrice = 1000;
uint256 _savedForBoardroomBase = stakedNfts.mul(nftPrice).mul(1e18);
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

PPI - Potential Precision Issue

Criticality	Minor / Informative
Location	TaxOracle.sol#L427
Status	Unresolved

Description

A variable is being cast from a higher precision data type to a lower precision data type. This can cause a loss of information and result in unexpected results. For example, if a value of 300 is cast to a uint8, the resulting value will be truncated to 44 because uint8 can only store values between 0 and 255. This can have unintended consequences if the variable is used in subsequent calculations or comparisons.

The code statement `uint144(myUSDBalance.mul(_amountIn).div(wbnbBalance))` might produce precision issues.

```
function consult(address _token, uint256 _amountIn) external view returns
(uint144 amountOut) {
    require(_token == address(myUSD), "token needs to be myUSD");
    uint256 myUSDBalance = myUSD.balanceOf(pair);
    uint256 wbnbBalance = wbnb.balanceOf(pair);
    return uint144(myUSDBalance.mul(_amountIn).div(wbnbBalance));
}
```

Recommendation

It is recommend reviewing the code to identify any variables that are being cast to lower precision data types. If possible, these variables should be modified to use higher precision data types. Alternatively, if it is necessary to use lower precision data types, it is important to carefully consider the implications of any casting operations and ensure that they do not result in loss of precision.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	Trasuryeasury.sol#L1292Boardroom.sol#L856,906,1005
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The variable `epoch` is a date and time representation, but the cotnract is utilizing the variable as a counter.

```
uint256 public epoch = 0;
epoch = epoch.add(1);
```

The variable `rewardEarned` depicts the withdrawable amount, not the earned rewards.

```
struct Boardseat {
    ...
    uint256 rewardEarned;
    ...
}

seat.rewardEarned = earned(board);

function claimReward() public updateReward(msg.sender) {
    uint256 reward = boards[msg.sender].rewardEarned;
    if (reward > 0) {
        ...
        boards[msg.sender].rewardEarned = 0;
        ...
    }
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MCN - Misleading Contract Naming

Criticality	Minor / Informative
Location	TaxOracle.sol Treasury.sol#L1312
Status	Unresolved

Description

Contract can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. Misleading contract names can lead to confusion, making the code more difficult to read and understand. Contracts can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve.

An oracle contract connects smart contracts with the outside world, by providing information from the world. The contract `TaxOracle` does not feed information from off chain sources.

```
address public myUsdOracle;

contract TaxOracle {...}
```

Recommendation

It's always a good practice for the contract to have accurate names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RC - Redundant Comments

Criticality	Minor / Informative
Location	Boardroom.sol Treasury.sol
Status	Unresolved

Description

The current implementation of the contract contains numerous comments that add little value to the codebase. For example, comments may repeat what is already expressed by the code, or provide unnecessary context to code blocks that are self-explanatory. This makes it harder to understand and modify the code, potentially leading to errors and security vulnerabilities.

```
// IBasisAsset(nft).operator() == address(this) &&
// user.amount = user.amount.add(_amount);
// nfts.safeTransferFrom(msg.sender, address(this), amount);
// uint256[] memory tokenId = nfts.walletOfOwner(msg.sender);
// user.amount = user.amount.sub(_amount);
// nfts.safeTransferFrom(address(this),msg.sender, amount);
// require(boards[msg.sender].epochTimerStart.add(withdrawLockupEpochs) <=
treasury.// epoch(), "Boardroom: still in withdraw lockup");
// require(boards[msg.sender].epochTimerStart.add(rewardLockupEpochs) <=
treasury.epoch(), "Boardroom: still in reward lockup");
```

Recommendation

It is recommended reviewing the code and removing any unnecessary comments that do not provide additional value to the codebase. Comments should be used sparingly, and only to provide additional context or information that is not immediately clear from the code. By reducing the number of comments in the code, it will become easier for developers to read and modify the code, ultimately improving the overall quality and security of the contract.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	Treasury.sol#L1533,1778
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

The comment about the variable `bootstrapEpochs` is misleading beacause it is mutable.

```
// First 21 epochs with 4.5% expansion
bootstrapEpochs = 21;
// 28 first epochs with 4.5% expansion
uint256 _savedForBoardroom;

function setBootstrap(uint256 _bootstrapEpochs, uint256
_bootstrapSupplyExpansionPercent) external onlyOperator {
    require(_bootstrapEpochs <= 120, "_bootstrapEpochs: out of range"); //
<= 1 month
    ...
    bootstrapEpochs = _bootstrapEpochs;
    ...
}
```

Recommendation

The team is advised to carefully review the comment in order to address these issues. To improve code readability, the team should use more specific and descriptive comment messages.

UCV - Unutilized Contract Variable

Criticality	Minor / Informative
Location	Treasury.sol#L1789
Status	Unresolved

Description

The contract contains variables that are calculated but not used in subsequent code blocks. This can lead to wasted computation and unnecessary gas costs, as well as potential security vulnerabilities.

The variable `_percentage` is initialized and calculated but it is not used in the contract implementation.

```
uint256 _mse = _calculateMaxSupplyExpansionPercent(myUsdSupply).mul(1e14);

uint256 _percentage = previousEpochMyUSDPrice.sub(myUsdPriceOne);
...
uint256 _mse = _calculateMaxSupplyExpansionPercent(myUsdSupply).mul(1e14);

if (_percentage > _mse) {
    _percentage = _mse;
}
```

Recommendation

It is recommended removing any variables that are calculated but not used in subsequent code blocks. By removing unused variables, the code will be more efficient and less prone to vulnerabilities.

RCS - Redundant Code Statement

Criticality	Minor / Informative
Location	Treasury.sol#L1758Boardroom.sol#L903
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `safeApprove` is called multiple times.

```
IERC20(myUsd).safeApprove(boardroom, 0);
IERC20(myUsd).safeApprove(boardroom, _amount);
```

The statement `if (board != address(0)){...}` is redundant, due to the fact the msg.sender will never be the zero address.

```
modifier updateReward(address board) {
    if (board != address(0)) {
        ...
    }
    _;
}

updateReward(msg.sender)
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

- The function `safeApprove` should be called only once.
- Redundant if statements should be removed.

TAP - Transfer Amount Prevalidation

Criticality	Minor / Informative
Location	Boardroom.sol Treasury.sol
Status	Unresolved

Description

The current implementation of the contract does not prevalidate whether sufficient tokens are available to perform transactions, which can lead to transaction failures or unexpected behavior. For example, if a user attempts to transfer tokens that are not available, the transaction will fail and potentially leave the contract in an inconsistent state.

```
myUSD.safeTransfer(msg.sender, reward);
IERC20(myUsd).safeTransfer(msg.sender, _myUsdAmount);
...
```

Recommendation

It is recommended to implement prevalidation checks before any transaction. To ensure that there are sufficient tokens available for any transaction. By performing prevalidation checks, the contract will be more reliable and less prone to errors or vulnerabilities.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MyUSDRewardPool.sol#L600,601
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public MyUSDPerSecond = 3805175038051750
uint256 public runningTime = 1095 days
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Treasury.sol#L857,1301,1499,1500,1501,1502,1503,1504,1549,1553,1557,1561,1566,1571,1578,1591,1599,1604,1609,1614,1622,1623,1624,1625,1637,1641,1645,1650,1656,1661,1682,1711,1825,1826,1827,1836,1840,1849,1850,1851TaxOracle.sol#L427,449,454,459MyBond.sol#L798Boardroom.sol#L921,922,923,939,943,949,1059
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
IERC20 USDC = IERC20(0x8AC76a51cc950d9822D68b83fE1Ad97B32Cd580d)
address _myUsd
address _myBond
address _nft
address _myUsdOracle
address _boardroom
uint256 _startTime
address _operator
uint256 _myUsdPriceCeiling
uint256 _maxSupplyExpansionPercent
uint256 _nftPercentDen
uint256 _nftPercent
uint256 _nftPrice

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Treasury.sol#L1301
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
IERC20 USDC = IERC20(0x8AC76a51cc950d9822D68b83fE1Ad97B32Cd580d)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	Treasury.sol#L1550,1554 Boardroom.sol#L940
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
operator = _operator
boardroom = _boardroom
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Treasury.sol#L1563,1568,1573,1601,1611,1617,1632,1638,1642,1647,1653,1658,1663Boardroom.sol#L945
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
myUsdPriceCeiling = _myUsdPriceCeiling
maxSupplyExpansionPercent = _maxSupplyExpansionPercent
nftPrice = _nftPrice
bondDepletionFloorPercent = _bondDepletionFloorPercent
maxDebtRatioPercent = _maxDebtRatioPercent
bootstrapEpochs = _bootstrapEpochs
daoFundSharedPercent = _daoFundSharedPercent
maxDiscountRate = _maxDiscountRate
maxPremiumRate = _maxPremiumRate
discountPercent = _discountPercent
premiumThreshold = _premiumThreshold
premiumPercent = _premiumPercent
mintingFactorForPayingDebt = _mintingFactorForPayingDebt
withdrawLockupEpochs = _withdrawLockupEpochs
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	Treasury.sol#L1579,1592,1765
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_index >= 0, "Index has to be higher than 0")
tierId >= 0
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Treasury.sol#L320,335,641,667,692,717,727,741,751,800,822,827,1127MyBond.sol#L601Boardroom.sol#L324,335,340,415,441,466,491,501,515,525
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
...
function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        // solhint-disable-next-line avoid-low-level-calls,
avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	Treasury.sol#L1416,1424
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint144 price
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Treasury.sol#L1801,1802,1805,1806
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 _seigniorage = stakedNfts.mul(nftPrice).mul(1e18).div(1000)
_savedForBoardroom =
_seigniorage.mul(seigniorageExpansionFloorPercent).mul(nftExpansionPercent).div(nftEx
pansionPercentDen).div(10000)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Treasury.sol#L1416,1424,1790Boardroom.sol#L803,820
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint144 price
uint256 _savedForBond
uint256 i
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Treasury.sol#L1506,1507,1509,1510,1550,1554,1558Boardroom.sol#L940
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
myUsd = _myUsd
myBond = _myBond
myUsdOracle = _myUsdOracle
boardroom = _boardroom
operator = _operator
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Treasury.sol#L621,768Boardroom.sol#L395,542
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	Treasury.sol#L6,30,97,311,342,591,780,852,1061,1099,1124,1143,1205,1216,1250,1271TaxOracle.sol#L7,403MyBond.sol#L3,28,105,319,623,663,730,768Boardroom.sol#L3,80,294,365,554,764,779
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity >=0.6.0 <0.8.0;
pragma solidity >=0.6.2 <0.8.0;
pragma solidity >=0.6.2;
pragma solidity 0.6.12;
pragma solidity ^0.6.0;

pragma solidity >=0.6.0 <0.8.0;
pragma solidity 0.6.12;

pragma solidity >=0.6.0 <0.8.0;
pragma solidity >=0.6.2 <0.8.0;
pragma solidity 0.6.12;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Treasury.sol#L1124,1250
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.6.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Treasury.sol#L1743,1750
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(myUsd).transfer(daoFund, _daoFundSharedAmount)
IERC20(myUsd).transfer(teamFund, _teamFundSharedAmount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	

	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
ContractGuard	Implementation			
	checkSameOriginReentranted	Internal		
	checkSameSenderReentranted	Internal		
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	walletOfOwner	External		-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	getApproved	External		-
	setApprovalForAll	External	✓	-
	isApprovedForAll	External		-
	safeTransferFrom	External	✓	-
ITreasury	Interface			

	epoch	External		-
	nextEpochPoint	External		-
	getMyUSDPrice	External		-
	buyBonds	External	✓	-
	redeemBonds	External	✓	-
ShareWrapper	Implementation			
	totalSupply	Public		-
	balanceOf	Public		-
	stake	Public	✓	-
	withdraw	Public	✓	-
	getStakedIds	Public		-
	getArrayIds	Public		-
Boardroom	Implementation	ShareWrapper, ContractGuard		
	initialize	Public	✓	notInitialized
	setOperator	External	✓	onlyOperator
	setLockUp	External	✓	onlyOperator
	updateBoardroom	External	✓	onlyOperator
	latestSnapshotIndex	Public		-
	getLatestSnapshot	Internal		
	getLastSnapshotIndexOf	Public		-

	getLastSnapshotOf	Internal		
	canWithdraw	External		-
	canClaimReward	External		-
	epoch	External		-
	nextEpochPoint	External		-
	getMyUSDPrice	External		-
	rewardPerShare	Public		-
	earned	Public		-
	stake	Public	✓	onlyOneBlock updateReward
	withdraw	Public	✓	onlyOneBlock boardExists updateReward
	exit	External	✓	-
	claimReward	Public	✓	updateReward
	allocateSeigniorage	External	✓	onlyOneBlock onlyOperator
	governanceRecoverUnsupported	External	✓	onlyOperator
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-

	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
ERC20	Implementation	Context, IERC20		
		Public	✓	-
	name	Public		-

	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_setupDecimals	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
Ownable	Implementation	Context		
		Internal	✓	

	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
Operator	Implementation	Context, Ownable		
		Internal	✓	
	operator	Public		-
	isOperator	Public		-
	transferOperator	Public	✓	onlyOwner
	_transferOperator	Internal	✓	
MyBond	Implementation	ERC20Burnable, Operator		
		Public	✓	ERC20
	mint	Public	✓	onlyOperator
	burn	Public	✓	-
	burnFrom	Public	✓	onlyOperator
	callget	Public	✓	onlyOwner
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		

	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-

	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
TaxOracle	Implementation	Ownable		
		Public	✓	-
	consult	External		-
	getMyUSDBalance	External		-
	getWbnbBalance	External		-
	getPrice	External		-
	setMyUSD	External	✓	onlyOwner
	setWbnb	External	✓	onlyOwner
	setPair	External	✓	onlyOwner
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner

	transferOwnership	Public	✓	onlyOwner
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
IERC20	Interface			

	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	getApproved	External		-
	setApprovalForAll	External	✓	-
	isApprovedForAll	External		-
	safeTransferFrom	External	✓	-
Address	Library			
	isContract	Internal		

	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-

	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-

	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Operator	Implementation	Context, Ownable		
		Internal	✓	
	operator	Public		-
	isOperator	Public		-
	transferOperator	Public	✓	onlyOwner
	_transferOperator	Internal	✓	
ContractGuard	Implementation			
	checkSameOriginReentranted	Internal		
	checkSameSenderReentranted	Internal		
Babylonian	Library			
	sqrt	Internal		
ReentrancyGuard	Implementation			
		Internal	✓	
IOracle	Interface			
	update	External	✓	-
	consult	External		-
	twap	External		-

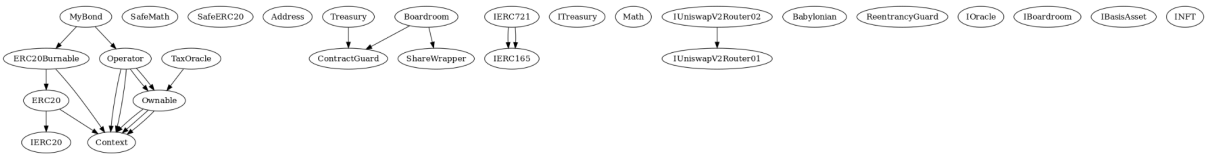
IBoardroom	Interface			
	balanceOf	External		-
	earned	External		-
	canWithdraw	External		-
	canClaimReward	External		-
	epoch	External		-
	nextEpochPoint	External		-
	getMyUSDPrice	External		-
	setOperator	External	✓	-
	setLockUp	External	✓	-
	stake	External	✓	-
	withdraw	External	✓	-
	exit	External	✓	-
	claimReward	External	✓	-
	allocateSeigniorage	External	✓	-
	governanceRecoverUnsupported	External	✓	-
IBasisAsset	Interface			
	mint	External	✓	-
	burn	External	✓	-
	burnFrom	External	✓	-
	isOperator	External	✓	-

	operator	External		-
	transferOperator	External	✓	-
INFT	Interface			
	price	External	✓	-
Treasury	Implementation	ContractGuard		
	isInitialized	Public		-
	nextEpochPoint	Public		-
	getMyUSDPrice	Public		-
	getMyUSDUpdatedPrice	Public		-
	getReserve	Public		-
	getBurnableMyUSDLeft	Public		-
	getRedeemableBonds	Public		-
	getBondDiscountRate	Public		-
	getBondPremiumRate	Public		-
	initialize	Public	✓	notInitialized
	setOperator	External	✓	onlyOperator
	setBoardroom	External	✓	onlyOperator
	setmyUsdOracle	External	✓	onlyOperator
	setmyUsdPriceCeiling	External	✓	onlyOperator
	setMaxSupplyExpansionPercents	External	✓	onlyOperator
	setNftInfo	External	✓	onlyOperator

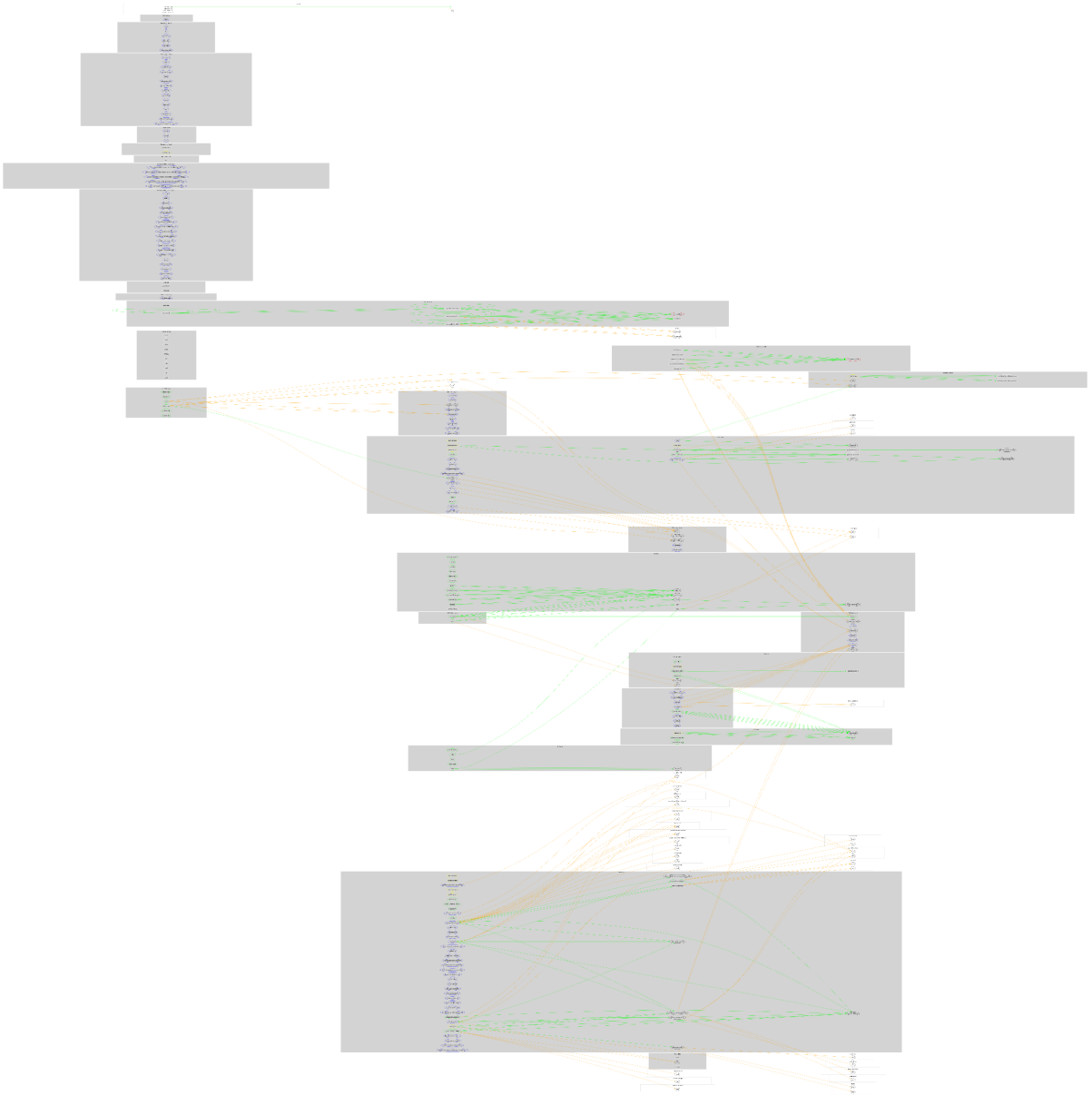
	setSupplyTiersEntry	External	✓	onlyOperator
	setMaxExpansionTiersEntry	External	✓	onlyOperator
	setBondDepletionFloorPercent	External	✓	onlyOperator
	setMaxSupplyContractionPercent	External	✓	onlyOperator
	setMaxDebtRatioPercent	External	✓	onlyOperator
	setBootstrap	External	✓	onlyOperator
	setExtraFunds	External	✓	onlyOperator
	setMaxDiscountRate	External	✓	onlyOperator
	setMaxPremiumRate	External	✓	onlyOperator
	setDiscountPercent	External	✓	onlyOperator
	setPremiumThreshold	External	✓	onlyOperator
	setPremiumPercent	External	✓	onlyOperator
	setMintingFactorForPayingDebt	External	✓	onlyOperator
	_updateMyUSDPrice	Internal	✓	
	getMyUSDCirculatingSupply	Public		-
	buyBonds	External	✓	onlyOneBlock checkCondition checkOperator
	redeemBonds	External	✓	onlyOneBlock checkCondition checkOperator
	_sendToBoardroom	Internal	✓	
	_calculateMaxSupplyExpansionPercent	Internal	✓	
	allocateSeigniorage	External	✓	onlyOneBlock checkCondition checkEpoch checkOperator
	governanceRecoverUnsupported	External	✓	onlyOperator

	boardroomSetOperator	External	✓	onlyOperator
	boardroomSetLockUp	External	✓	onlyOperator
	boardroomAllocateSeigniorage	External	✓	onlyOperator
	boardroomGovernanceRecoverUnsupported	External	✓	onlyOperator

Inheritance Graph



Flow Graph



Summary

MyBricks contract implements a token, utility, financial and rewards mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io