Cyberscope

Audit Report

Dual Pools Trend Token

May 2023

Github	https://github.com/JavisJL/dualpools/tree/main/TrendTokenAudit
Commit	9e2f82407dc61f75e8a95c350ec1690b67b97f9c
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/JavisJL/dualpools
Commit	9e2f82407dc61f75e8a95c350ec1690b67b97f9c

Audit Updates

Initial Audit	16 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/xdp/v1/trendToken.pdf
Corrected Phase 2	24 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/xdp/v2/trendToken.pdf
Corrected Phase 3	28 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/xdp/v3/trendToken.pdf
Corrected Phase 4	17 May 2023

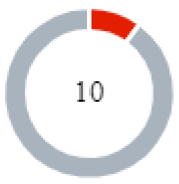
Source Files

Filename	SHA256
Address.sol	df780035071456dfc4b41b4647f1ba1e69d a5e97eff3b0c3c1f2ed5eb4c24994
AggregatorV2V3Interface.sol	a5523f8072d46e9a121390beefd22cf9b78 d00efec1e1f1274c010cacb019495
CompStorageTT.sol	e19ef5d663608b1dfa8132d32ff8e0911a0e 0f373aff70745cdbd9bf004723bd
CompTT.sol	46800ee02f4577e21967d9b748eab84be0 7d1248cac4312951a7d0858d9fe396
DualPool.sol	ddf24f1548fab966f15e4fb3c02ed950fa3f4 a1221c86787b5716a377d2dde62
DualPoolStorage.sol	d0b5fdd0579d88997b23be8a436bddbf0d bdc69429a7a96c1295d83a5e73a306
ERC20.sol	6b52089f84c5f7e6fbf7f2219ebe31f43e10 4ce89b44b28a72a4371c19a847e6
ERC20Detailed.sol	edbc6746642b1fdb6066c5dacd1c3756f0 4c4af57521594fbcb8864c61e5b648
IChainlinkOracle.sol	27cac8821c279a09cdf6b3e0e1985867db 49ef35cce6763df8154daafb77a31e
ICompDP.sol	1fbb60793e8bf070e6e70f05d3e9c67557e c73aa637630146f4cdc4206073b48
ICompTT.sol	b71da5599b0e6bbe9fc1d163880a8c3d02 90f9d315c97ac212f99ecde715e818
IERC20.sol	23221a896472eeee23d71500d71f40bcce 31112b9198389310d2e7ff7d0be093

IIncentiveModelSimple.sol	a3a4ee3d50e418aa6163a3a63c55e45194 1a0f931848fb259158b41cc489f520
IncentiveModelSimple.sol	cccacfd0a45806c02ad9f09fece60dc71b5 785d83dc2d172405b50e1a9e510a1
ITrendToken.sol	7eac00453f65769353394e73abfc46ef366 ec79bd640069669afd08438abbe00
ITrendTokenTkn.sol	b3286f0e63b4fb4f77cc7d012fff28c06b63 90686542f9c107a65e3b9549137e
IVBep20.sol	4388fcfcdf67909073a1af7353687e653b91 80653281418aced5a607afcc78b8
IVBNB.sol	53487fc7336311df84bdb8faa5d7999ae61 b48b9f72f462ceae53dc34f631435
IXTT.sol	36ff8e43a69fc4b0da21352c652aee0b733 c841be27fcf3035d9583ea6519ac7
Lens.sol	dfd684164358d48b86653c429aa994f52f1 5b244656a1bed2c3da8b30082e9f6
Lib.sol	38b01b1d3513df71ab674cd6061e93c1d4 34f5f57efecab71c2131d07a8965e9
SafeERC20.sol	02ac28e67c7239a7a8137b864ef01063b4 b649f8c4609edfb62f1829a8ee9185
SafeMath.sol	4a47d15402f20ec26b0fe15d61f4f6e946e7 949b7beaa6398957b5cadee42931
SignedSafeMath.sol	533257d850b02a32792adfa2e02af99e926 a1b08af501b37eae82cc174b9c06a
TrendToken.sol	67e8ab020cbaf5b9a4fbd9b83867b3a42f8 b8fab815932b7721afc3d5e529a32
TrendTokenStorage.sol	999b4f49e5523522c58649f04bf832d93bf dc51dd6923111cb862c8bbf00a3a5

TrendTokenTkn.sol	0022a6b8e6d71e032073f4f6aad22a294d19098e72d51c097bf82e179a7f44a8
UniTT.sol	57442216e7dc8993ab5fe99cb5699b23865cd42fd0f8b983e87746de31e32003
XTTgov.sol	11160e5e1becd7e1f6f43af23b635a24592ade0b1627b4d79252ff64ed6e292b

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	9

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	1	0	0
●	Medium	0	0	0	0
●	Minor / Informative	5	4	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🔴	UIA	Unsafe Indexing Assumption	Acknowledged
⚪	UTFO	Update Token Fee Optimization	Unresolved
⚪	MU	Modifiers Usage	Unresolved
⚪	MT	Mints Tokens	Acknowledged
⚪	IDI	Immutable Declaration Improvement	Unresolved
⚪	L02	State Variables could be Declared Constant	Acknowledged
⚪	L04	Conformance to Solidity Naming Conventions	Acknowledged
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L16	Validate Variable Setters	Unresolved
⚪	L19	Stable Compiler Version	Acknowledged

UIA - Unsafe Indexing Assumption

Criticality	Critical
Status	Acknowledged

Description

The contract assumes that the `desiredAllocations` indexes are identical to the market's tokens indexes. This assumption could be broken if the `desiredAllocations` or `market's tokens` change without updating the corresponding structures. As a result, the entire contract will yield to an unexpected state.

```
desiredAllocations = _allocations;
emit SetDesiredAllocationsFresh(getMarkets(), oldAllocations,
desiredAllocations);

//...

if (IVBep20(dTokens[i]) == dTokensInOut[0]) {
    tokenEquityInOut[0] = conVals[i].add(colVals[i]);
    desiredAllocations[0] = desiredAllocations[i];
}
```

Recommendation

The team is advised to carefully investigate the circumstances where the indexes could be diverse. A recommended way could be to allow only synchronized modifications of these properties.

Team Update

Trading bot will ensure proper index when changing order of markets (tokens) or adjusting desired allocations.

We acknowledge the risk and make sure to update appropriately, but think this is more of a "medium" or even "minor" risk severity.

UTFO - Update Token Fee Optimization

Criticality	Minor / Informative
Location	IncentiveModelSimple.sol#L201
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 contract is utilizing the method `underlyingSupported` which performs a iterational cost of O(n) to verify whether the token is valid. The contract could leverage the `feePerToken` mapping to avoid this computational overhead.

```
function _updateFeePerToken(IERC20 underlying, uint feeOrReward) external {
    require(msg.sender == admin,"!admin");
    require(feeOrReward > 0 && feeOrReward <= 0.05e18,"max 5%, min>0");
    if (!underlyingSupported(underlying)) {
        allUnderlying.push(underlying);
    }
    feePerToken[address(underlying)] = feeOrReward;
}
```

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. For instance, using the mapping data structure, the time complexity will be degreased from O(n) to O(1).

```
if (feePerToken[address(underlying)] == 0) {
  allUnderlying.push(underlying);
}
```

MU - Modifiers Usage

Criticality	Minor / Informative
Location	TrendToken.sol#L869,876,976,1054
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

The `deadlineExceeded` method is utilized in multiple functions.

```
function redeem(IERC20 _redeemBep20, uint _redeemAmt, uint
_maxTrendTokenIn, uint _deadline) external nonReentrant {
    ...
    deadlineExceeded(_deadline);
}

function depositBNB(uint _minTrendTokenOut, uint _deadline, address
payable _referrer) external nonReentrant payable {
    ...
    deadlineExceeded(_deadline);
}

function deposit(IERC20 _depositBep20, uint _sellAmtBEP20, uint
_minTrendTokenOut, address payable _referrer, uint _deadline)
external nonReentrant {
    ...
    deadlineExceeded(_deadline);
}

function executeTrade(IERC20[] memory tokenInOut, uint sellAmt,
uint _minOut, uint _deadline) internal pausedTrendToken {
    ...
    deadlineExceeded(_deadline);
    ...
}
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

MT - Mints Tokens

Criticality	Minor / Informative
Location	DualPool.sol#L317
Status	Acknowledged

Description

When the authorized addresses have the necessary actions unlocked, the manager role has the authority to arbitrarily mint tokens. If all the conditions are met, the managers can invoke the `_supplyCollateral` function, leading to potential inflation of tokens within the contract.

```
function _adjustCollateral(IERC20 _bep20, uint _supplyAmt, uint
_redeemAmt) onlyManager requireUnlocked external {
    IVBep20 dToken = dTokenSupportedRequire(_bep20);
    if (_supplyAmt > 0) {
        collateralSupply(_bep20,dToken, _supplyAmt);
    } else if (_redeemAmt > 0) {
        collateralRedeem(_bep20,dToken,_redeemAmt);
    }
}
```

Recommendation

The team should carefully manage the private keys and the implementation of the trading bot 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 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.

Team Update

This only affects supplying and redeeming supplied assets to the lending/borrowing protocol Dual Pools (fork of Venus). In addition to the ability to mint tokens (supply and redeem collateral) being restricted to the manager, it also requires these actions to be "unlocked" by the tradingBot.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	TrendTokenTkn.sol#L21
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
minter
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	TrendTokenStorage.sol#L106,113,127,134,140,146,152,159,165,171,178,185,193,207,215,223,233 DualPoolStorage.sol#L16,22,28,34,40 CompStorageTT.sol#L13,18,23,28,38,87,93,123,129
Status	Acknowledged

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.

```
bool internal _notEntered
uint public contractFactor = 1e18
uint public maxSupply = 10000e18
address public manager
address public tradingBot
address payable public feeRecipient
IIncentiveModelSimple public incentiveModel
bool public trendTokenPaused = false
uint public referralReward = 0.40e18
uint public performanceFee = 0.10e18
uint public trendTokenRedeemBurn = 0.50e18
uint public maxDisableTokenValue = 1e18
bool locked = true
ITrendTokenTkn public trendToken

...
```

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.

Team Update

State variables cannot be declared constant. All variables in storage (i.e Trend TokenStorage.sol) are changeable in other files (i.e TrendToken.sol), therefore, they cannot be declared constant

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TrendTokenTkn.sol#L31,35 TrendTokenStorage.sol#L106 TrendToken.sol#L104,155,173,204,235,250,261,274,286,296,307,317,335,346,358,372,402,412,428,439,458,467,480,495,511,520,537,577,606,669,699,745,782,807,828,867,876,898,917,939,976,1040,1062,1071,1101,1151 Lib.sol#L27,32 DualPool.sol#L51,70,82,95,105,116,128,144 CompTT.sol#L218,227,239,250,261,273,297,335,374,394,403,413,423,434,445
Status	Acknowledged

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.

```
address _recipient
uint256 _amount
bool internal _notEntered
address _owner
uint _deadline
address _compTT

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	TrendToken.sol#L263
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.

```
referralReward = _referralReward
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CompTT.sol#L253,306
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.

```
lockedWallet = _newWallet
pauseGuardian = newPauseGuardian
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TrendTokenTkn.sol#L2 TrendTokenStorage.sol#L2 TrendToken.sol#L2Lib.sol#L2 ITrendTokenTkn.sol#L2 ITrendToken.sol#L2 ICompTT.sol#L2ICompDP.sol#L2 DualPoolStorage.sol#L2 DualPool.sol#L2 CompTT.sol#L2 CompStorageTT.sol#L2
Status	Acknowledged

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 ^0.5.16;
```

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.

Team Update

We will make sure to deploy using 0.5.16 compiler

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	toPayable	Internal		
	sendValue	Internal	✓	
AggregatorV2V3Interface	Interface			
	latestAnswer	External		-
	latestTimestamp	External		-
	latestRound	External		-
	getAnswer	External		-
	getTimestamp	External		-
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-
UnitrollerAdminStorage	Implementation			

ComptrollerStorage	Implementation	UnitrollerAdminStorage		
CompTT	Implementation	ComptrollerStorage		
		Public	✓	-
	ensureAdmin	Private		
	ensureNonzeroAddress	Private		
	depositOrRedeemAllowed	External		onlyProtocolAllowed
	tradeAllowed	External		onlyProtocolAllowed
	getBlockNumber	External		-
	getXVSAddress	External		-
	returnDToken	External		onlyProtocolAllowed
	trendTokenIsListed	External		-
	trendTokenIsActive	External		-
	trendTokenIsTrade	External		-
	trendTokenAllowedDualPools	External		-
	trendTokenMaxTradeFee	External		-
	trendTokenMaxPerformanceFee	External		-
	trendTokenMaxDisableValue	External		-
	_become	External	✓	requireUnlocked
	_setProtocolPaused	External	✓	validPauseState
	_updateLockedState	External	✓	-
	_updateLockedWallet	External	✓	-
	_setMintPaused	External	✓	validPauseState

	_setPriceOracle	External	✓	requireUnlocked
	_setPauseGuardian	External	✓	requireUnlocked
	supportTokenFresh	Internal	✓	
	_supportToken	External	✓	requireUnlocked
	trendTokenSupported	Internal		
	_supportTrendTokenFresh	Internal	✓	
	_supportTrendToken	External	✓	requireUnlocked
	_newIsActive	External	✓	requireUnlocked onlySupportedTrendTokens
	_newIsTrade	External	✓	requireUnlocked onlySupportedTrendTokens
	_newAllowedDualPools	External	✓	requireUnlocked onlySupportedTrendTokens
	_newMaxTradeFee	External	✓	requireUnlocked onlySupportedTrendTokens
	_newMaxPerformanceFee	External	✓	requireUnlocked onlySupportedTrendTokens
	_newMaxDisableValue	External	✓	requireUnlocked onlySupportedTrendTokens
DualPoolIntegration	Implementation	DualPoolStorage		
		Public	✓	-
		External	Payable	-

	screenshot	Internal		
	getMarkets	Internal		
	priceBEP20	Internal		
	exchangeVBEP20	Internal		
	enableCol	Internal	✓	
	disableCol	Internal	✓	
	tokenEntered	Internal		
	collateralSupply	Internal	✓	
	collateralRedeem	Internal	✓	
DualPoolStorage	Implementation			
ERC20	Implementation	IERC20		
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

	_approve	Internal	✓	
	_burnFrom	Internal	✓	
ERC20Detailed	Implementation	IERC20		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
IOracle	Interface			
	getUnderlyingPrice	External		-
	getFeed	External		-
	getChainlinkPrice	External		-
ICompDP	Interface			
	enterMarkets	External	✓	-
	claimXDP	External	✓	-
	venusAccrued	External		-
	getAssetsIn	External		-
	markets	External		-
	getAccountLiquidity	External		-
	closeFactorMantissa	External		-
	exitMarket	External	✓	-
	getHypotheticalAccountLiquidity	External		-
	checkMembership	External		-

	iUSDaddress	External		-
ICompTT	Implementation			
	oracle	External		-
	protocolPaused	External		-
	depositOrRedeemAllowed	External		-
	tradeAllowed	External		-
	returnDToken	External		-
	trendTokenIsListed	External		-
	trendTokenIsActive	External		-
	trendTokenIsTrade	External		-
	trendTokenAllowedDualPools	External		-
	trendTokenMaxTradeFee	External		-
	trendTokenMaxPerformanceFee	External		-
	trendTokenMaxDisableValue	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

IIncentiveModelSimple	Interface			
	protocolFeeTrade	External		-
	protocolFeeDeposit	External		-
	protocolFeeRedeem	External		-
	isIncentiveModel	External		-
	totalDepositFee	External		-
	totalRedeemFee	External		-
	valueOutAfterSell	External		-
	valueOutAfterBuy	External		-
IncentiveModelSimple	Implementation	IIncentiveModelSimple		
		Public	✓	-
	_updateTradeFeeDiscounts	External	✓	-
	_updateDepositFee	External	✓	-
	_updateRedeemFee	External	✓	-
	_updateProtocolFeeTrade	External	✓	-
	feeDiscount	Public		-
	underlyingSupported	Internal		
	_updateFeePerToken	External	✓	-
	returnFeePerToken	Internal		
	returnFeePerTokenExt	External		-
	depositRewardOrFee	Internal		
	redeemRewardOrFee	Internal		
	totalDepositFee	External		-

	totalRedeemFee	External		-
	valueOutAfterSell	External		-
	valueOutAfterBuy	External		-
ITrendToken	Interface			
	incentiveModel	External		-
	storedEquity	External		-
	trendToken	External		-
	performanceFee	External		-
	desiredAllocations	External		-
	lastRebalance	External		-
	isTrendToken	External		-
	compDP	External		-
	dBNB	External		-
	priceExt	External		-
	trendTokenToUSDext	External		-
	trendTokenOutExternal	External		-
	trendTokenInExternal	External		-
	tradeInfoExt	External		-
ITrendTokenTkn	Interface			
	mint	External	✓	-
	name	External		-
	burn	External	✓	-
	totalSupply	External		-

	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transfersFrom	External	✓	-
IVBep20	Interface			
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	mint	External	✓	-
	repayBorrow	External	✓	-
	redeemUnderlying	External	✓	-
	exchangeRateStored	External		-
	borrowBalanceCurrent	External	✓	-
	borrow	External	✓	-
	getCash	External		-
	getAccountSnapshot	External		-
	borrowBalanceStored	External		-
	totalBorrowsCurrent	External	✓	-
	accrueInterest	External	✓	-
	amountsOut	External		-
	swapExactTokensForTokens	External	✓	-
	underlying	External		-
IVBNB	Interface			

	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	mint	External	Payable	-
	repayBorrow	External	Payable	-
	redeemUnderlying	External	✓	-
	exchangeRateStored	External		-
	borrowBalanceCurrent	External	✓	-
	borrow	External	✓	-
	getAccountSnapshot	External		-
	borrowBalanceStored	External		-
	totalBorrowsCurrent	External	✓	-
	swapExactETHForTokens	External	Payable	-
IXTT	Interface			
	burn	External	✓	-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transfersFrom	External	✓	-
TrendLens	Implementation			
		Public	✓	-
	trendTokenData	Public		-

	trendTokenDataAll	Public		-
	swapPairsOld	External		-
	trendTokenPortfolio	Public		-
	removeUnderlyingFromPortfolio	Public		-
	portfolioAddTrendToken	Public		-
	swapPairs	External		-
	trendTokenOutAndFee	Public		-
	underlyingInAndFee	Public		-
	depositAmountsAndFee	Public		-
	trendTokenInAndFee	Public		-
	underlyingOutAndFee	Public		-
	redeemAmountsAndFees	Public		-
	balance	Public		-
	walletBalance	Public		-
	amountsAndFees	External		-
	amountsAndFeesTradeHelper	Internal		
	amountsAndFeesTrade	External		-
Lib	Library			
	pathGenerator2	Internal		
	getValue	Internal		
	getAssetAmt	Internal		
SafeERC20	Library			
	safeTransfer	Internal	✓	

	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	callOptionalReturn	Private	✓	
SafeMath	Library			
	add	Internal		
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
SignedSafeMath	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
TrendToken	Implementation	DualPoolIntegration,		

		TrendTokenStorage		
		Public	✓	DualPoolIntegration
		External	Payable	-
	onlyModifiers	Internal		
	deadlineExceeded	Internal		
	_updateCompAndModels	External	✓	onlyManager requireUnlocked
	_updateManagerRecipientAndBot	External	✓	onlyManager requireUnlocked
	_newPerformanceFee	External	✓	onlyManager requireUnlocked
	_updateTrendTokenBurn	External	✓	onlyManager
	_setReferralReward	External	✓	onlyManager
	_maxDisableValue	External	✓	onlyManager requireUnlocked
	_setMaxSupply	External	✓	onlyManager
	setContractFactor	Internal	✓	
	_setContractFactor	External	✓	onlyManager
	_adjustCollateral	External	✓	onlyManager requireUnlocked
	_updateLocked	External	✓	onlyTradingBot
	_depositsDisabled	External	✓	onlyTradingBot
	_pauseTrendToken	External	✓	onlyTradingBot
	dTokenSupportedRequire	Internal		
	_setDesiredAllocationsFresh	Internal	✓	
	_setDesiredAllocations	External	✓	onlyTradingBot

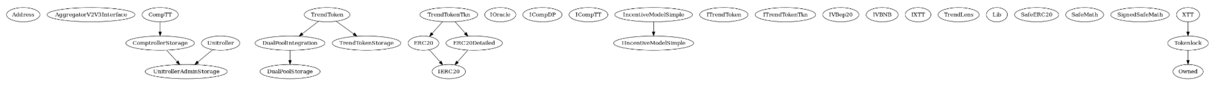
	_enableTokens	External	✓	onlyTradingBot
	checkActiveToken	Internal		
	_disableToken	External	✓	onlyTradingBot
	_redeemPerformanceFee	External	✓	onlyManager
	_reduceTrendTokenReservesToRecipient	External	✓	onlyManager
	_redeemXDPtoRecipient	External	✓	onlyManager
	priceExt	External		-
	trendTokenToUSDext	External		-
	trendTokenOutExternal	External		-
	trendTokenInExternal	External		-
	storedEquityExternal	External		-
	tokenInfoExternal	External		-
	tradeInfoExt	External		-
	calculatePerformanceFee	Internal		
	sendPerformanceFee	Internal	✓	
	balanceXDP	Internal		
	contractBal	Internal		
	storedEquity	Internal		
	trendTokenToUSD	Internal		
	trendTokenToUSD	Internal		
	tokenInfo	Internal		
	tokenEquityVal	Internal		
	distributeReferralReward	Internal	✓	
	trendTokenOutCalculations	Internal		

	depositFresh	Internal	✓	pausedTrendToken
	depositBNB	External	Payable	nonReentrant
	deposit	External	✓	nonReentrant
	sendUnderlyingOut	Internal	✓	
	trendTokenInCalculations	Internal		
	redeemFresh	Internal	✓	pausedTrendToken
	redeem	External	✓	nonReentrant
	tradeInfo	Internal		
	executeTrade	Internal	✓	pausedTrendToken
	swapExactTokensForTokens	External	✓	nonReentrant
	swapExactETHForTokens	External	Payable	nonReentrant
	returnUnderlying	Internal		
	singleSupplyAndRedeemRebalance	Internal	✓	
	publicSupplyAndRedeemRebalance	External	✓	pausedTrendToken
TrendTokenStorage	Implementation			
TrendTokenTkn	Implementation	ERC20, ERC20Detailed		
		Public	✓	ERC20Detailed
	mint	External	✓	requireMinter
	burn	External	✓	-
	transfersFrom	External	✓	-

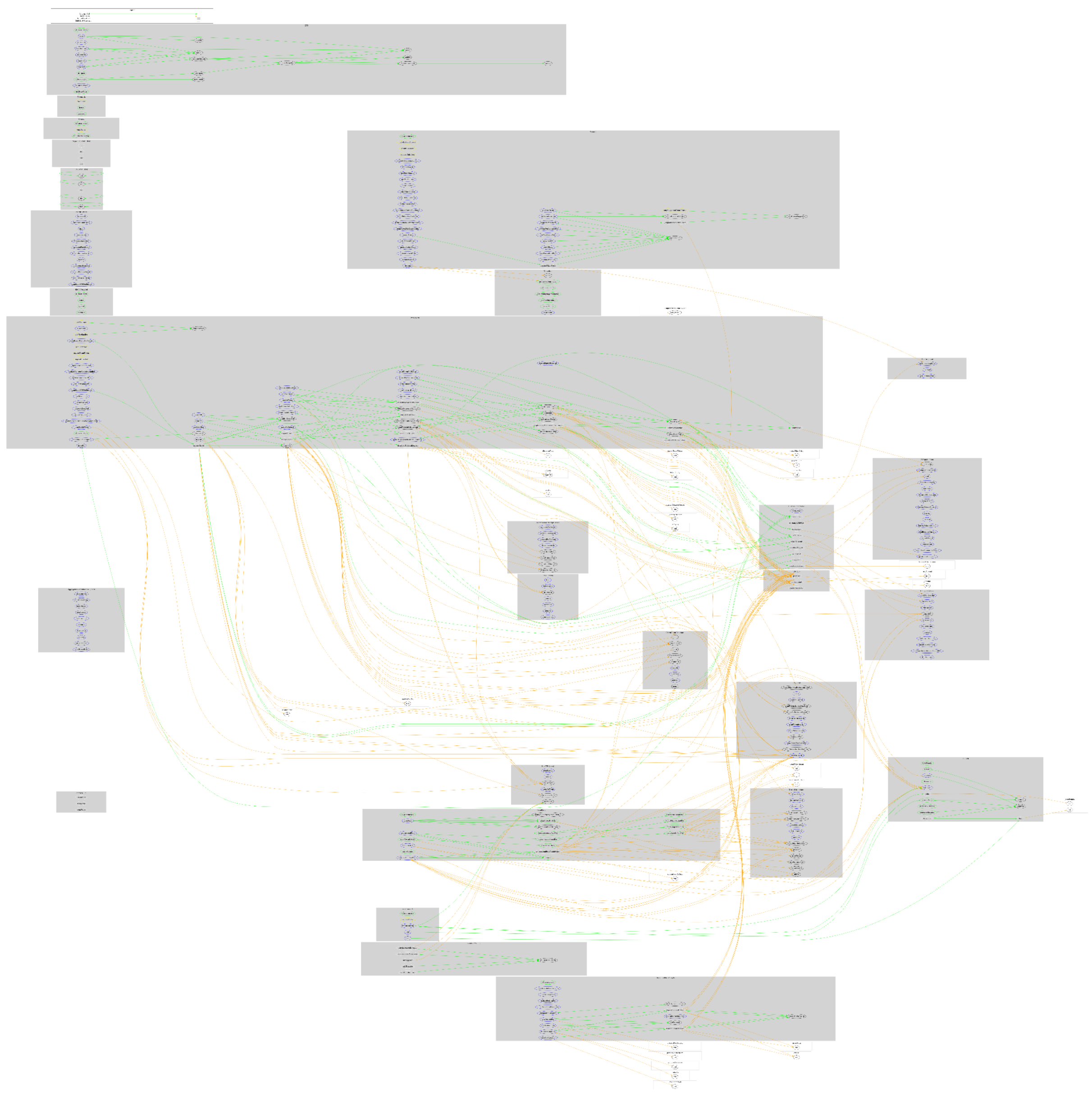
Unitroller	Implementation	UnitrollerAdminStorage		
		Public	✓	-
	_setPendingImplementation	Public	✓	-
	_acceptImplementation	Public	✓	-
	_setPendingAdmin	Public	✓	-
	_acceptAdmin	Public	✓	-
		External	Payable	-
Owned	Implementation			
		Public	✓	-
	transferOwnership	Public	✓	onlyOwner
Tokenlock	Implementation	Owned		
	freeze	Public	✓	onlyOwner
	unfreeze	Public	✓	onlyOwner
XTT	Implementation	Tokenlock		
		Public	✓	-
	burn	External	✓	-
	allowance	External		-
	approve	External	✓	validLock
	balanceOf	External		-
	transfer	External	✓	validLock

	transferFrom	External	✓	validLock
	delegate	Public	✓	validLock
	delegateBySig	Public	✓	validLock
	getCurrentVotes	External		-
	getPriorVotes	Public		-
	_delegate	Internal	✓	
	_transferTokens	Internal	✓	
	_moveDelegates	Internal	✓	
	_writeCheckpoint	Internal	✓	
	safe32	Internal		
	safe96	Internal		
	add96	Internal		
	sub96	Internal		
	sub256	Internal		
	getChainId	Internal		

Inheritance Graph



Flow Graph



Summary

Dual Pools contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

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