Cyberscope

Audit Report

Dual Pools Trend Token

April 2023

Github https://github.com/JavisJL/dualpools/tree/main/TrendTokenAudit

Commit ee03d8eeb0b412dd944ea4aeb61fbf26aa2aca5e

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/JavisJL/dualpools
Commit	ee03d8eeb0b412dd944ea4aeb61fbf26aa2aca5e

Audit Updates

Initial Audit	16 Apr 2023

Source Files

Filename	SHA256
AggregatorV2V3Interface.sol	a5523f8072d46e9a121390beefd22cf9b78d00efec1e1f1274c010cacb019495
CompStorageTT.sol	3e0959f600bd1eb11cb54a9a98df75244cbce6ba91f38fcd2c8971c6680a5242
CompTT.sol	7b277ae21ad474553412b708bfdf7873d5bb5231d1fbf39cf74d621e4b70ecb7
DualPool.sol	47b77b66e41e241a618bb086fe24dc7083d73ef61054828f83cf323d7b1f2c12
DualPoolStorage.sol	d0b5fdd0579d88997b23be8a436bddbf0dbdc69429a7a96c1295d83a5e73a306
ERC20.sol	6b52089f84c5f7e6fbf7f2219ebe31f43e104ce89b44b28a72a4371c19a847e6
ERC20Detailed.sol	edbc6746642b1fdb6066c5dacd1c3756f04c4af57521594fbcb8864c61e5b648
IChainlinkOracle.sol	27cac8821c279a09cdf6b3e0e1985867db49ef35cce6763df8154daafb77a31e
ICompDP.sol	1fbb60793e8bf070e6e70f05d3e9c67557ec73aa637630146f4cdc4206073b48
ICompTT.sol	b71da5599b0e6bbe9fc1d163880a8c3d0290f9d315c97ac212f99ecde715e818
IERC20.sol	23221a896472eeee23d71500d71f40bcce31112b9198389310d2e7ff7d0be093
IIncentiveModelSimple.sol	a3a4ee3d50e418aa6163a3a63c55e451941a0f931848fb259158b41cc489f520

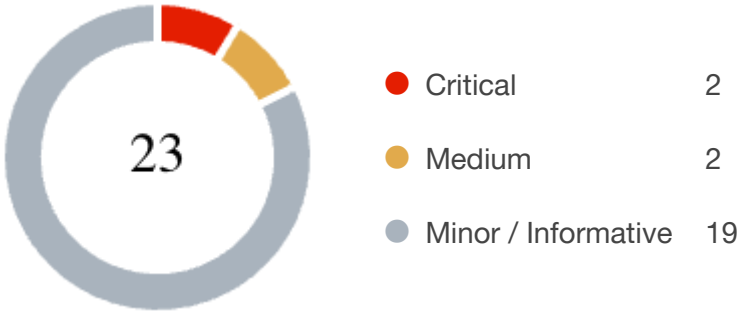
IncentiveModelSimple.sol	a514d85096a1e7c7ef209478aba46d6f4c1bac7dfd404f63893b037e9746011d
IPancakeRouter.sol	c61657a8afd44ac0167d429ff2a1dd6390631da7006e3d60e92af45be968c3cc
ITrendToken.sol	23405749543ca2ef8bd05b3ff2599c0c026d519f6ba4f47569fcb047f80d1941
ITrendTokenTkn.sol	b3286f0e63b4fb4f77cc7d012fff28c06b6390686542f9c107a65e3b9549137e
IVBep20.sol	4388fcfcdf67909073a1af7353687e653b9180653281418aced5a607afcc78b8
IVBNB.sol	53487fc7336311df84bdb8faa5d7999ae61b48b9f72f462ceae53dc34f631435
IXTT.sol	36ff8e43a69fc4b0da21352c652aee0b733c841be27fcf3035d9583ea6519ac7
Lib.sol	d422b07cb39ad6c13941d82d5d9dc524583a85640fb441afb997fc829dfb49a7
SafeMath.sol	4a47d15402f20ec26b0fe15d61f4f6e946e7949b7beaa6398957b5cadee42931
SignedSafeMath.sol	533257d850b02a32792adfa2e02af99e926a1b08af501b37eae82cc174b9c06a
TrendToken.sol	0f41568f1c520e727717aff33f022fa1a61acbcfd1ee3371b4479759135add5a
TrendTokenStorage.sol	2c3736dd21656a822cac8a4c82656aac3c1dfb823481e15ff2ecd6c8acf21899
TrendTokenTkn.sol	7113aa9235ca431ea743ba70f36e00c68cea3de39c8ac935986923d9244445cf
UniTT.sol	57442216e7dc8993ab5fe99cb5699b23865cd42fd0f8b983e87746de31e32003

XTTgov.sol	11160e5e1becd7e1f6f43af23b635a24592ade0b1627b4d79252ff64ed6e292b

Audit Comments

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 following findings, the contract cannot be assumed that it is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure the contract adheres to established best practices and security measures. It is recommended that the team should reconsider the expected behavior and re-implement most of the source code parts. The code's readability should also be improved by simplifying function definitions and using descriptive variable names.

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	2	0	0	0
Minor / Informative	19	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PSU	Potential Subtraction Underflow	Unresolved
●	UIA	Unsafe Indexing Assumption	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	DSD	Data Storage Duplication	Unresolved
●	RCE	Redundant Code Execution	Unresolved
●	MT	Mints Tokens	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	REE	Redundant Equity Evaluation	Unresolved
●	TPOE	Transitive Property of Equality	Unresolved
●	MUT	Mutating Unsupported Tokens	Unresolved
●	RDC	Redundant Duplicate Checks	Unresolved
●	RG	Redundant Getters	Unresolved
●	UM	Unused modifiers	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved

●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PSU - Potential Subtraction Underflow

Criticality	Critical
Location	TrendToken.sol#L713
Status	Unresolved

Description

The contract subtracts the `priorAllocation` from the `desiredAllocation`. If the desiredAllocation is less than the `priorAllocation`, then the contract will underflow. There is not qarantee from the contract's business logic that the `desiredAllocations` will always be greater than the `priorAllocation`.

```
uint tokenEquity = conVals[i].add(colVals[i]);
uint priorAllocation = Lib.getAssetAmt(tokenEquity,netEquity);
priorDelta = int(desiredAllocations[i]).sub(int(priorAllocation));
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

UIA - Unsafe Indexing Assumption

Criticality	Critical
Status	Unresolved

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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Medium
Location	TrendToken.sol#L856
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
function deposit(IERC20 _depositBep20, uint _sellAmtBEP20, address
payable _referrer) external nonReentrant {
    _depositBep20.transferFrom(msg.sender, address(this),
_sellAmtBEP20);
    depositFresh(_depositBep20, _sellAmtBEP20,_referrer);
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

DSD - Data Storage Duplication

Criticality	Medium
Status	Unresolved

Description

The contract adds values to arrays without checking if they already exist. As a result, the data structures could contain duplicate information. The duplication may harm the proper operation of the ecosystem.

```
function supportTokenFresh(address underlying, address dToken)
internal returns(uint) {
    ensureAdmin();
    require(oracle.getUnderlyingPrice(dToken) != 0,"no price in
Oracle");
    tokenToDToken[underlying] = dToken;
    supportedTokens.push(underlying);
}
```

Recommendation

The team is advised to prevent duplicate values in the arrays when it is not required by the business logic.

RCE - Redundant Code Execution

Criticality	Minor / Informative
Status	Unresolved

Description

The contract performs repetitive checks for variables that are not related to the loop context. For instance, in the following example, the `netEquity` is not related to the loop. These statements decrease the readability and maintainability of the source code.

```
uint netEquity) = storedEquity();

for (uint i=0; i < dTokens.length; i++) {

    if (address(dToken) == dTokens[i]) {

        uint tokenEquity = conVals[i].add(colVals[i]);

        if (netEquity>0 && tokenEquity > 0) {

            ...
        }
    }
}
```

Recommendation

The team is advised to remove expressions that are not related to the loops outside the repetitive context.

MT - Mints Tokens

Criticality	Minor / Informative
Location	DualPool.sol#L147
Status	Unresolved

Description

The trading bot role has the authority to arbitrary mint tokens. The role may take advantage of it by calling the `_supplyCollateral` function. As a result, the contract tokens will be highly inflated.

```
function _supplyCollateral(IERC20 _depositBep20, uint supplyAmt)
onlyTradingBot external {
    IVBep20 dToken = dTokenSupportedRequire(_depositBep20);
    collateralSupply(_depositBep20,dToken, supplyAmt);
}
```

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 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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

In the following example, if the token equity value is more than the required the contract will revert to the "cannot disable BNB" message that is not related to the actual issue.

```
require(_bep20 != wbnb && tokenEquityVal(dToken) <
maxDisableTokenValue,"cannot disable BNB");
```

Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

REE - Redundant Equity Evaluation

Criticality	Minor / Informative
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 `storedEquity()` is a heavy expression in terms of gas consumption since it evaluates all the market's tokens. The contract is using this method to retrieve information only from one market token.

```
function tokenEquityVal(IVBep20 _dToken) internal view
returns(uint) {
    (address[] memory dTokens,,
    uint[] memory conVals,
    uint[] memory colVals,,) = storedEquity();

    for (uint i=0; i < dTokens.length; i++) {
        if (IVBep20(dTokens[i]) == _dToken) {
            return conVals[i].add(colVals[i]);
        }
    }
}
```

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.

TPOE - Transitive Property of Equality

Criticality	Minor / Informative
Location	TrendToken.sol#L321
Status	Unresolved

Description

The contract checks if each unsigned integer of an array is less than `1e18` and later it checks if the sum of the array is less than `1e18`. According to the Transitive Property of Equality, the first check could be removed.

```
function _setDesiredAllocationsFresh(uint[] memory _allocations)
internal {
    require(_allocations.length == getMarkets().length, "!length");
    uint allocationTotal = 0;
    for (uint i=0; i<_allocations.length; i++) {
        uint allocation = _allocations[i];
        require(allocation <= 1e18, "max allocation 100%.");
        allocationTotal += allocation;
    }
    require(allocationTotal == 1e18, "allocation != 100%");
}
```

Recommendation

The team is advised to check if the conditions are following the Transitive Property of Equality and remove the unnecessary checks.

MUT - Mutating Unsupported Tokens

Criticality	Minor / Informative
Status	Unresolved

Description

The contract allows the admin to set values for trend token addresses that are not supported by the platform. This may lead the contract to an unexpected state.

```
_newIsActive(...);
_newIsTrade(...);
_newAllowedDualPools(...);
_newMaxTradeFee(...);
_newMaxPerformanceFee(...);
_newMaxDisableValue(...);
```

Recommendation

The team is advised to allow state modifications only when the pre-requirements are fulfilled. For instance, the contract should modify the trend token properties if the trend token already exists.

RDC - Redundant Duplicate Checks

Criticality	Minor / Informative
Location	CompTT.sol#L336
Status	Unresolved

Description

The contract performs the same check twice. This duplication unnecessarily increases gas consumption and decreases the readability of the contract.

```
trendToken.isTrendToken();

//...

require(trendToken.isTrendToken(),"not a trend token");
```

Recommendation

The team is advised to remove the duplicate checks in the contract. It is recommended to keep the checks with descriptive messages in case of failure.

RG - Redundant Getters

Criticality	Minor / Informative
Status	Unresolved

Description

The contract uses getter methods for variables that are already public.

```
function getAllTrendTokens() external view returns (ITrendToken[]
memory) {
    return allTrendTokens;
}
```

Recommendation

The team is advised to remove the getter methods or change the visibility of the underneath variables to private.

UM - Unused modifiers

Criticality	Minor / Informative
Status	Unresolved

Description

The contract contains the same validators both as modifiers and functions. The code repetition produced from these methods unnecessarily increases the code size and decreases the readability.

The following example depicts one of these cases.

```
modifier onlyListedTrendTokens(ITrendToken trendToken) {
    require(trendTokens[address(trendToken)].isListed, "venus
market is not listed");
    _;
}

...

require(trendTokens[trendToken].isListed, "trend token not
listed");

...

function trendTokenIsListed(address trendToken) external view
returns(bool) {
    return trendTokens[address(trendToken)].isListed;
}
```

Recommendation

The team is advised to use a single point of truth instead of repeating the same functionality.

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#L59,66,80,87,93,99,105,112,118,124,131,138,145,159,167,175,181,198DualPoolStorage.sol#L16,22,28,34,40CompStorageTT.sol#L13,18,23,28,38,87,93
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.

```
bool internal _notEntered
uint public contractFactor = 0.90e18
uint public maxSupply = 1000e18
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 accruedXDPtoFeeRecipient = 0.50e18
uint public maxDisableTokenValue = 1e18
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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TrendTokenTkn.sol#L35,39TrendTokenStorage.sol#L59,153TrendToken.sol#L92,148,179,210,225,236,249,259,267,276,287,297,309,341,351,367,378,392,411,421,433,442,452,474,497,506,523,563,592,662,692,735,764,789,810,848,856,874,893,915,952,1012,1069,1119Lib.sol#L27,36,41,56DualPool.sol#L70,89,101,114,124,135,147,163CompTT.sol#L218,227,236,248,272,309,337,359,368,378,388,399,410
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.

```
address _recipient
uint256 _amount
bool internal _notEntered
bool public constant isTrendToken = true
address _owner
address _compDP

...
```

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#L214,227,238,260
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.

```
performanceFee = _performanceFee
trendTokenRedeemBurn = _trendTokenRedeemBurn
referralReward = _referralReward
maxSupply = _maxSupply
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Lib.sol#L19,27,36,41,48,56DualPool.sol#L46,57,70,80,89,101,114,124,135,147,163CompTT.sol#L76
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 pathGenerator2(address coinIn, address coinOut) internal
pure returns(address[] memory) {
        address[] memory path = new address[](2);
        path[0] = address(coinIn);
        path[1] = address(coinOut);
        return path;
    }

function pathGenerator3(address _coinIn, address _interRoute1,
address _coinOut) internal pure returns(address[] memory) {
        address[] memory path = new address[](3);
        path[0] = address(_coinIn);
        path[1] = address(_interRoute1);
        path[2] = address(_coinOut);
        return path;
    }

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	TrendToken.sol#L666
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.

```
uint mintTrendToken
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	TrendToken.sol#L975
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint[] memory desiredAllocations = new uint[](2)
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CompTT.sol#L281
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.

```
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
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 ^0.5.0;
```

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	TrendToken.sol#L414,462,776,777,857,883,1031
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.

```
trendToken.transfer(feeRecipient,redeemAmtTrendToken)
xdp.transfer(feeRecipient,transferAmount)
_token.transfer(msg.sender, distributeAmount)
_token.transfer(referrer, distributeAmount)
_depositBep20.transferFrom(msg.sender, address(this),
_sellAmtBEP20)
_underlying.transfer(msg.sender, _amount)
tokenInOut[0].transferFrom(msg.sender, address(this), sellAmt)
```

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**
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		
	adminOrInitializing	Internal		
	depositOrRedeemAllowed	External		onlyProtocolAllowed
	tradeAllowed	External		onlyProtocolAllowed
	getAllTrendTokens	External		-
	getBlockNumber	External		-
	getXVSAddress	External		-
	returnDToken	External		onlyProtocolAllowed
	trendTokenIsListed	External		-
	trendTokenIsActive	External		-
	trendTokenIsTrade	External		-
	trendTokenAllowedDualPools	External		-
	trendTokenMaxTradeFee	External		-
	trendTokenMaxPerformanceFee	External		-
	trendTokenMaxDisableValue	External		-
	_become	External	✓	-
	_setProtocolPaused	External	✓	validPauseState
	_setMintPaused	External	✓	validPauseState
	_setPriceOracle	External	✓	-
	_setPauseGuardian	External	✓	-
	supportTokenFresh	Internal	✓	
	_supportToken	External	✓	-

	_supportTrendTokenFresh	Internal	✓	
	_supportTrendToken	External	✓	-
	_newIsActive	External	✓	-
	_newIsTrade	External	✓	-
	_newAllowedDualPools	External	✓	-
	_newMaxTradeFee	External	✓	-
	_newMaxPerformanceFee	External	✓	-
	_newMaxDisableValue	External	✓	-
DualPoolIntegration	Implementation	DualPoolStorage		
		Public	✓	-
		External	Payable	-
	_newCompDPinternal	Internal	✓	
	amtAccruedXDP	Internal		
	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	✓	-
	feeDiscount	Public		-
	feePerToken	Public		-
	depositRewardOrFee	Public		-
	redeemRewardOrFee	Public		-
	totalDepositFee	Public		-
	totalRedeemFee	Public		-
	valueOutAfterSell	External		-
	valueOutAfterBuy	External		-
IPancakeRouter	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		-
ITrendToken	Interface			
	incentiveModel	External		-
	storedEquity	External		-
	trendToken	External		-
	performanceFee	External		-
	lastRebalance	External		-
	isTrendToken	External		-
	dBNB	External		-

	priceExt	External		-
	trendTokenToUSD	External		-
	trendTokenOutExternal	External		-
	trendTokenInExternal	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	✓	-
Lib	Library			
	pathGenerator2	Internal		
	pathGenerator3	Internal		
	getValue	Internal		
	getAssetAmt	Internal		
	min	Internal		
	countValueArray	Internal		
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		
	_updateContracts	External	✓	onlyManager
	_updateAddresses	External	✓	onlyManager
	_newPerformanceFee	External	✓	onlyManager
	_updateFeeDistribution	External	✓	onlyManager
	_setReferralReward	External	✓	onlyManager

	_maxDisableValue	External	✓	onlyManager
	_setMaxSupply	External	✓	onlyManager
	setContractFactor	Internal	✓	
	_setContractFactor	External	✓	onlyManager
	_depositsDisabled	External	✓	onlyTradingBot
	_pauseTrendToken	External	✓	onlyTradingBot
	dTokenSupportedRequire	Internal		
	_setDesiredAllocationsFresh	Internal	✓	
	_setDesiredAllocations	External	✓	onlyTradingBot
	_enableTokens	External	✓	onlyTradingBot
	checkActiveToken	Internal		
	_disableToken	External	✓	onlyTradingBot
	_swapXDPforBNB	External	✓	onlyTradingBot
	_reduceTrendTokenReservesToRecipient	External	✓	onlyManager
	_distributePerformanceFee	External	✓	onlyManager
	_supplyCollateral	External	✓	onlyTradingBot
	_redeemCollateral	External	✓	onlyTradingBot
	_redeemXDP	External	✓	onlyTradingBot
	priceExt	External		-
	balanceXDPext	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

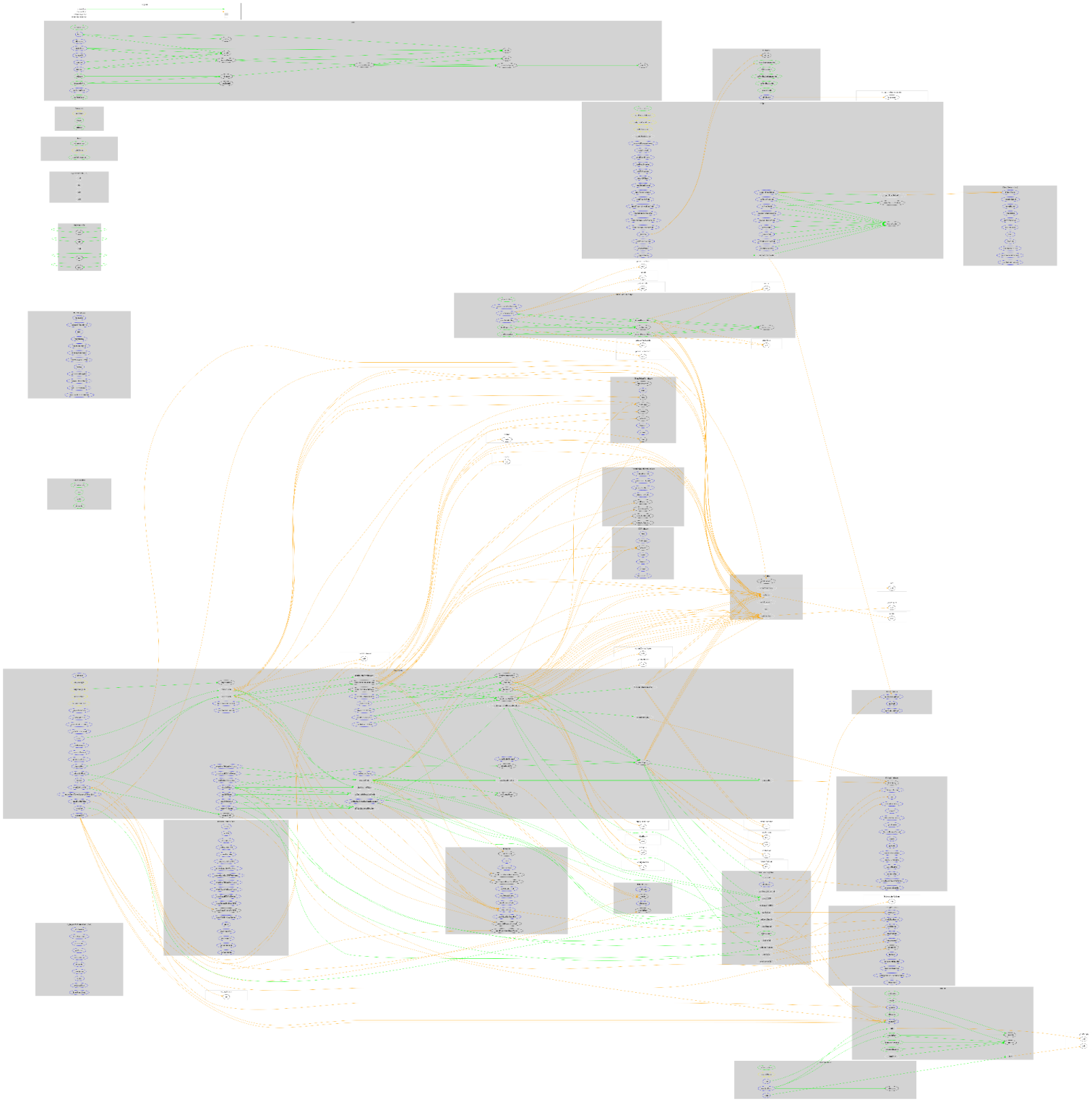
	swapExactTokensForTokens	External	✓	nonReentrant
	swapExactETHForTokens	External	Payable	nonReentrant
	returnUnderlying	Internal		
	singleSupplyAndRedeemRebalance	Internal	✓	
	publicSupplyAndRedeemRebalance	External	✓	pausedTrendToken
TrendTokenStorage	Implementation			
TrendTokenTkn	Implementation	ERC20, 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

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