Cyberscope

Audit Report

Golduck Reward Pool

March 2023

Network ETH

Address 0x6D2c7B38557B39776FbB82f593e82C3eB0d6E159

Audited by © cyberscope

Table of Contents

Review

Explorer	https://etherscan.io/address/0x6d2c7b38557b39776fbb82f593e82c3eb0d6e159
Address	0x6D2c7B38557B39776FbB82f593e82C3eB0d6E159

Audit Updates

Initial Audit	27 Mar 2023

Source Files

Filename	SHA256
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	da66c17044345dc892d85bd7ddc9745d25df0b3dacfba8f84eb87c60d6e40fe3
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	cd823c76cbf5f5b6ef1bda565d58be66c843c37707cd93eb8fb5425deebd6756
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	c05b019a0b3bee8f3fac2da7c929f7d665b97d6d046aa35126615fff11205119
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	35fb271561f3dc72e91b3a42c6e40c2bb2e788cd8ca58014ac43f6198b8d32ca
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/security/ReentrancyGuard.sol	aa73590d5265031c5bb64b5c0e7f84c44cf5f8539e6d8606b763adac784e8b2e
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Address.sol	1e0922f6c0bf6b1b8b4d480dcabb691b1359195a297bde6dc5172e79f3a1f826
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a
contracts/RPD_V1/interfaces/IRewardDistributor.sol	babacd04aa2e629f7c51178dee6587babac3189ee6adc8590bd1aa8867c39e1a

contracts/RPD_V1/interfaces/IRewardPool.sol	d3b22fba6f8d5355828bd836f690ad84f d35792537067ed8273db91153cb5d98
contracts/RPD_V1/interfaces/IUniswapV2Factory.sol	888ac9f4ca776e2ddc256fac9c5d8c5f5 097eb957ea6b2ce2266a8a11f122f68
contracts/RPD_V1/interfaces/IUniswapV2Pair.sol	31f6be26511b5fd3471887e8a21a584e2 51805939bb3de59f8321bf988611187
contracts/RPD_V1/interfaces/IUniswapV2Router01.sol	8d20af7f337be186f0b58251c3e544bf1b dee32b0cb9c7322c5b351e6139d950
contracts/RPD_V1/interfaces/IUniswapV2Router02.sol	977cff5641af6f116f92cbce502e0fd8646 5c617e44499805a69b42188aef0c7
contracts/RPD_V1/library/IterableMapping.sol	a382a5c3181b2438e8e5deb0120e38a4 0476615c25eb7a28dcba42512b333f3d
contracts/RPD_V1/library/SafeMathInt.sol	db3630dda161bfe2172e052bcb7800a2 5076e9a29c8b15da07980038f66d9bcf
contracts/RPD_V1/library/SafeMathUint.sol	2b2864c9f4e90c87c7dc8b3ad5d85710 11536de3ad6a09d0cbf1e96dfb0129cf
contracts/RPD_V1/RewardDistributor.sol	9b7fbebe4d1a09aede9053fd90d2cac7f 7a840f9354756b5be846a70bdbd0b05
contracts/RPD_V1/RewardPool.sol	72f1245298d9c9ec27eb77da0a7217aa 9499bef20fa3b4b7116809f981409b0d

Introduction

The RewardPool contract is a custom implementation of a DividendTracker that distributes rewards to tokens instead of token holders.

Roles

Owner

The owner has authority over the following functions:

- `function pause()`
- `function unpause()`
- `function recoverLeftOverCoinAmount(uint256 amount)`
- `function recoverLeftOverToken(address token,uint256 amount)`
- `function setBalanceForBuyback(uint256 newMinValue,uint256 newMaxValue)`
- `function setMinimumTokenBalanceForRewards(address reward,uint256 newValue)`
- `function setDistributeShare(address rewardToken,uint256 newShare)`
- `function setBuyBackWait(uint256 newBuyBackWait)`
- `function createRewardDistributor(address _rewardToken, uint256 _distributeShare, uint256 _claimWait, uint256 _minimumTokenBalanceForRewards)`
- `function removeRewardToken(address rewardToken)`
- `function migarateDistributor(address _oldRewardToken, address _newRewardToken, uint256 _distributeShare, uint256 _claimWait, uint256 _minimumTokenBalanceForRewards)`
- `function setRewardActiveStatus(address rewardAsset,bool status)`
- `function generateBuyBack(uint256 buyBackAmount)`
- `function setPairAndRouter(address _uniswapV2Router,address _uniswapV2Pair)`
- `function updateGasForProcessing(uint256 newValue)`

- `function updateClaimWait(address rewardToken,uint256 claimWait)`
- `function excludeFromRewards(address account)`
- `function includeInRewards(address account)`

Operator

The operator has authority over the following functions:

- `function setBalance(address account, uint256 newBalance)`

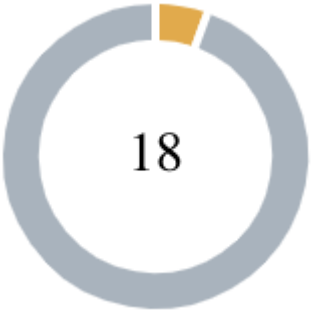
User

The user can interact with the following functions:

- `function validateDistributeShare(uint256 newShare)`
- `function getBuyBackLimit(uint256 currentBalance)`
- `function generateBuyBackForOpen()`
- `function getClaimWait(address rewardToken)`
- `function getTotalRewardsDistributed(address reward)`
- `function getAccountRewardsInfo(address reward,address _account)`
- `function getAccountRewardsInfoAtIndex(address reward,uint256 index)`
- `function removedTokenRewardClaim(address rewardToken)`
- `function singleRewardClaimByUser(address rewardToken)`
- `function multipleRewardClaimByUser()`
- `function getLastProcessedIndex(address rewardToken)`
- `function totalHolderSupply(address rewardToken)`
- `function getNumberOfTokenHolders(address reward)`
- `function autoDistribute(address rewardToken)`
- `function updateBalance()`
- `function updateBalanceForAll(address[] memory accounts)`
- `function withdrawableRewardOf(address reward,address account)`
- `function rewardOf(address reward,address account)`
- `function withdrawnRewardOf(address reward,address user)`
- `function getRewardsDistributor(address rewardAsset)`

- `function getRewardDistributorInfo(address rewardAsset)`
- `function getTotalNumberofRewardsDistributor()`
- `function getPoolStatus(address rewardAsset)`
- `function rewardsDistributorAt(uint8 index)`
- `function getAllRewardsDistributor()`
- `function getMinmumAndMaximumBuyback()`
- `function rewardInfo(address rewardToken)`
- `function distributeInfo(address reward,address user)`
- `function coinBalance()`
- `function isExcludedFromReward(address account)`
- `function rewardAssetAt(uint8 index)`

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	17	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor	SRD	Stops Reward Distribution	Unresolved
Minor	SRW	Stops Reward Withdrawal	Unresolved
Minor	DRT	Duplicate Reward Tokens	Unresolved
Minor	MEM	Misleading Error Messages	Unresolved
Minor	MMN	Misleading Method Naming	Unresolved
Minor	OCTD	Transfers Contract's Tokens	Unresolved
Minor	TUU	Time Units Usage	Unresolved
Minor	AAO	Accumulated Amount Overflow	Unresolved
Minor	RSML	Redundant SafeMath Library	Unresolved
Minor	IDI	Immutable Declaration Improvement	Unresolved
Minor	L02	State Variables could be Declared Constant	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L05	Unused State Variable	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PTAI - Potential Transfer Amount Inconsistency

Criticality	Medium
Location	contracts/RPD_V1/RewardPool.sol#L643
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

The `swapAndSendReward` function transfers the reward amount to the distributor and updates the `magnifiedRewardPerShare` and `totalRewardsDistributed` variables based on that amount. If the `rewardAsset` token charges fees as part of the transfer flow, then the amount that is used to update the variables mentioned above will be incorrect.

```
bool success =
IERC20(rewardAsset).transfer(_rewardInfo[rewardAsset].rewardDistributor, rewards);
```

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
```

SRD - Stops Reward Distribution

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardDistributor.sol#L252,273
Status	Unresolved

Description

The reward distribution can be delayed for a huge period of time. This can be achieved by setting the `buyBackWait` to a high value. As a result, the rewards will not be able to be distributed.

```
require(lastBuyBackTimestamp.add(buyBackWait) < block.timestamp, "RewardPool:
buybackclaim still not over");
```

Recommendation

The contract could embody a check for not allowing setting the `buyBackWait` more than a reasonable amount. 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.

SRW - Stops Reward Withdrawal

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L432,441,459
Status	Unresolved

Description

The reward withdrawal can be stopped for any asset. This can be achieved by setting either the

- `isActive` property of an asset to false.
- `claimWait` property of an asset to a high value.

As a result, the assets will not be able to claim their rewards.

```
require(_rewardInfo[rewardToken].isActive, "RewardPool: Pool is not active");

if(_rewardInfo[_rewardAsset[i]].isActive) {
    _withdrawRewardsOfUser(_rewardAsset[i],user,false);
}
...
if(lastClaimTime > block.timestamp)  {
    return false;
}
```

Recommendation

The contract could embody a check for not allowing setting the `claimWait` more than a reasonable amount, or being able to change the `isActive` property of an asset. 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.

DRT - Duplicate Reward Tokens

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L195
Status	Unresolved

Description

The function `removeRewardToken` removes a token from the reward distribution. The contract overrides the token's value with the last element of the `rewardAsset` array. The contract does not remove the last element after this action. As a result, the last element of the array will have more than one instance in the array.

```
uint8 index = _rewardInfo[rewardToken].index;
_rewardAsset[index] = _rewardAsset[totalRewardDistributor - 1];
_rewardInfo[_rewardAsset[totalRewardDistributor - 1]].index = index;
```

Recommendation

The team is advised to remove the last element of the `rewardAsset` array after its value is assigned to the token's index that is removed. This way, the contract will ensure that the array values are unique.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L426
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.

The error message provided is incorrect. If the `isRemoved` variable is false, then it means that the token is still active. However, the contract provides the exact opposite message, which can lead to confusion.

```
require(_rewardInfo[rewardToken].isRemoved, "RewardPool: Pool is not active");
```

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.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L204
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The function has a wrong spelling, which can lead to confusion.

```
function migarateDistributor
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code. Most likely, the function's name was supposed to be `migrateDistributor`.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L119
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `recoverLeftOverToken` function.

```
function recoverLeftOverToken(address token,uint256 amount) external
onlyOwner {
    IERC20(token).transfer(owner(),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.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L92
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
buyBackWait = 86400;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` , `weeks,` and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardDistributor.sol#L38
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
totalRewardsDistributed = totalRewardsDistributed.add(amount);
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardDistributor.solcontracts/RPD_V1/RewardPool.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change at https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardDistributor.sol#L22
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`.

```
rewardToken
```

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	contracts/RPD_V1/RewardPool.sol#L44
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 private swapping
```

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	contracts/RPD_V1/RewardPool.sol#L34,35,36,46,60,87,154,155,156,157,205,206,207,208,209,297,362contracts/RPD_V1/RewardDistributor.sol#L30
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 public constant deadWallet =
0x000000000000000000000000000000000000dEaD
uint256 internal constant magnitude = 2**128
uint256 internal constant distributeSharePrecision = 100

...
```

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	contracts/RPD_V1/RewardPool.sol#L44
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.

```
bool private swapping
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L127,150
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.

```
minimumCoinBalanceForBuyback = newMinValue
buyBackWait = newBuyBackWait
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L137,261,286,329,339,440,519,543,692
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.

```
uint8 i
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	contracts/RPD_V1/RewardPool.sol#L91,299
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.

```
nativeAsset = _nativeAsset
uniswapV2Pair = _uniswapV2Pair
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/RPD_V1/RewardPool.sol#L120contracts/RPD_V1/RewardDistributor.sol#L36,48
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(token).transfer(owner(),amount)
rewardToken.transfer(account, amount)
```

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**
IRewardDistributor	Interface			
	totalRewardsDistributed	External		-
	distributeReward	External	✓	-
IRewardPool	Interface			
	rewardOf	External		-
	withdrawnRewardOf	External		-
	setBalance	External	✓	-
RewardDistributor	Implementation	Ownable		
		Public	✓	-
	setRewardPool	External	✓	onlyOwner
	distributeReward	External	✓	onlyOperator

	recoverLeftOverBNB	External	✓	onlyOwner
	recoverLeftOverToken	External	✓	onlyOwner
	rewardOf	External		-
	withdrawnRewardOf	External		-
RewardPool	Implementation	Initializable, PausableUpgradeable, OwnableUpgradeable, ReentrancyGuard		
		External	Payable	-
	initialize	Public	✓	initializer
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	recoverLeftOverCoinAmount	External	✓	onlyOwner
	recoverLeftOverToken	External	✓	onlyOwner
	setBalanceForBuyback	External	✓	onlyOwner
	setMinimumTokenBalanceForRewards	External	✓	onlyOwner

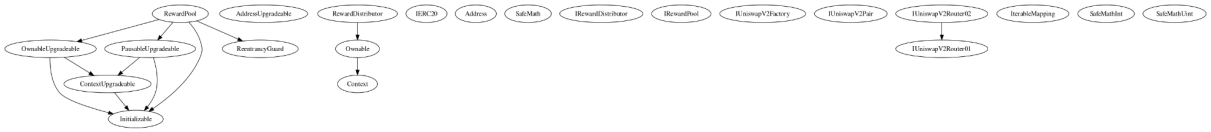
	validateDistributeShare	Public		-
	setDistributeShare	External	✓	onlyOwner
	setBuyBackWait	External	✓	onlyOwner
	createRewardDistributor	External	✓	onlyOwner
	removeRewardToken	External	✓	onlyOwner
	migarateDistributor	External	✓	onlyOwner
	setRewardActiveStatus	External	✓	onlyOwner
	getBuyBackLimit	Internal		
	generateBuyBackForOpen	External	✓	whenNotPaused nonReentrant
	generateBuyBack	External	✓	whenNotPaused onlyOwner nonReentrant
	setPairAndRouter	Public	✓	onlyOwner
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalRewardsDistributed	External		-

	_excludedFromRewards	Internal	✓	
	excludeFromRewards	External	✓	onlyOwner
	includeInRewards	External	✓	onlyOwner
	getAccountRewardsInfo	Public		-
	accumulativeRewardOf	Internal		
	getAccountRewardsInfoAtIndex	Public		-
	removedTokenRewardClaim	External	✓	whenNotPaused nonReentrant
	singleRewardClaimByUser	External	✓	whenNotPaused nonReentrant
	multipleRewardClaimByUser	External	✓	whenNotPaused nonReentrant
	getLastProcessedIndex	External		-
	totalHolderSupply	External		-
	getNumberOfTokenHolders	Public		-
	canAutoClaim	Private		
	autoDistribute	External	✓	-

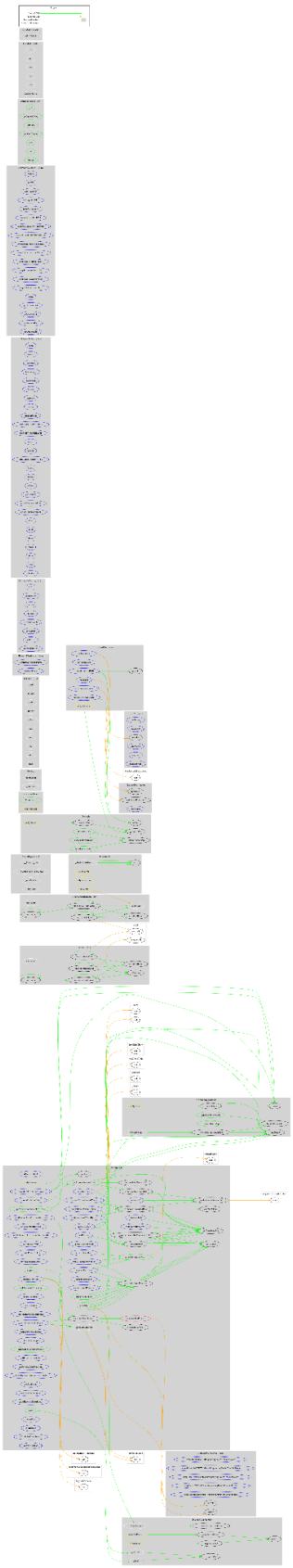
	setBalance	External	✓	onlyOperator
	updateBalance	External	✓	whenNotPaused
	updateBalanceForAll	External	✓	whenNotPaused
	_updateBalanceForRemomvedToken	Internal	✓	
	_updateBalance	Internal	✓	
	_setBalance	Internal	✓	
	_withdrawRewardsOfUser	Internal	✓	
	withdrawableRewardOf	Public		-
	rewardOf	External		-
	_withdrawableRewardOf	Internal		
	withdrawnRewardOf	External		-
	swapCoinForReward	Private	✓	
	swapAndSendReward	Internal	✓	
	distributeRewards	Internal	✓	
	getRewardsDistributor	External		-
	getRewardDistributorInfo	External		-
	getTotalNumberofRewardsDistributor	External		-

	getPoolStatus	External		-
	rewardsDistributorAt	External		-
	getAllRewardsDistributor	External		-
	getDistributeSlot	Internal		
	getMinmumAndMaximumBuyback	External		-
	rewardInfo	External		-
	distributeInfo	External		-
	coinBalance	External		-
	isExcludedFromReward	External		-
	rewardAssetAt	External		-

Inheritance Graph



Flow Graph



Summary

Golduck contract implements a 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