Cyberscope

Audit Report

MeeKyunDosa

March 2023

Network	BSC
Address	0xbCA00C0E5D1f62f6877179398Ca58b3d9A30Bf17
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Analysis 4
Diagnostics 5
DDP - Decimal Division Precision 7
Description 7
Recommendation 7
TSD - Total Supply Diversion 8
Description 8
Recommendation 8
RSML - Redundant SafeMath Library 9
Description 9
Recommendation 9
RSK - Redundant Storage Keyword 10
Description 10
Recommendation 10
L02 - State Variables could be Declared Constant 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 13
L05 - Unused State Variable 14
Description 14
Recommendation 14
L07 - Missing Events Arithmetic 15
Description 15
Recommendation 15
L09 - Dead Code Elimination 16
Description 16
Recommendation 16
L12 - Using Variables before Declaration 17
Description 17
Recommendation 17
L13 - Divide before Multiply Operation 18
Description 18

Recommendation 18
L14 - Uninitialized Variables in Local Scope 19
Description 19
Recommendation 19
L15 - Local Scope Variable Shadowing 20
Description 20
Recommendation 20
L16 - Validate Variable Setters 21
Description 21
Recommendation 21
L20 - Succeeded Transfer Check 22
Description 22
Recommendation 22
Functions Analysis 23
Inheritance Graph 33
Flow Graph 34
Summary 35
Disclaimer 36
About Cyberscope 37

Review

Contract Name	MeeKyunDosa
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	10 runs
Explorer	https://bscscan.com/address/0xbca00c0e5d1f62f6877179398ca58b3d9a30bf17
Address	0xbca00c0e5d1f62f6877179398ca58b3d9a30bf17
Network	BSC
Symbol	MKD
Decimals	18
Total Supply	10.000.000

Audit Updates

Initial Audit	06 Mar 2023 https://github.com/cyberscope-io/audits/tree/main/1-mkd/v1/audit.pdf
Corrected Phase 2	07 Mar 2023 https://github.com/cyberscope-io/audits/tree/main/1-mkd/v2/audit.pdf
Corrected Phase 3	27 Mar 2023

Source Files

Filename	SHA256
MeeKyunDosa.sol	030392f8a92bcfa0e78b49dc0d542c05fe057a0f4f66ec4c78eb48075cfa68ee

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	DDP	Decimal Division Precision	Unresolved
●	TSD	Total Supply Diversion	Acknowledged
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/MKD.sol#L1010
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 totalBuySell = buyAmount.add(sellAmount);
uint256 swapAmountBought = contractTokenBalance.mul(buyAmount).div(totalBuySell);
uint256 swapAmountSold = contractTokenBalance.mul(sellAmount).div(totalBuySell);
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

TSD - Total Supply Diversion

Criticality	Minor / Informative
Location	contracts/MKD.sol#L1030
Status	Acknowledged

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

The contract deducts a certain percentage of the transfer amount as dead fees and burns those tokens by transferring them to the `DEAD` address. The amount of tokens burned is then subtracted from the total supply. As a result, the sum of balances will be equal to the total supply.

```
if (deadFees > 0) {
    burntokens = amount.mul(deadFees) / 100;
    super._transfer(from, DEAD, burntokens);
    _totalSupply = _totalSupply.sub(burntokens);
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/MeeKyunDosa.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 unnecessarily the gas consumption.

```
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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/MeeKyunDosa.sol#L1572,1576,1583,1587
Status	Unresolved

Description

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

```
Map storage map
Map storage map
Map storage map
Map storage map
```

Recommendation

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

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/MeeKyunDosa.sol#L556
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.

```
address public DEAD = 0x000000000000000000000000000000000000dEaD
```

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/MeeKyunDosa.sol#L37,38,50,106,360,442,496,500,504,508,556,569,630,755,774,840,1216,1330
Status	Unresolved

Description

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

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

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

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
uint256 internal _totalSupply
function WETH() external pure returns (address);
uint256 internal constant magnitude = 2**128
address _owner
address public DEAD = 0x000000000000000000000000000000000000dEaD
address payable public BBPWallet
event updateMarketingWallet(address wallet);
uint256 GWEI

...
```

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/MeeKyunDosa.sol#L307
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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

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/MeeKyunDosa.sol#L757,763,770,783,788,1124
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.

```
gasPriceLimit = GWEI * 1 gwei
cooldowntimer = value
maxWallet = value
maxTX = value
swapTokensAtAmount = amount * (10**18)
buyAmount = buyAmount.sub(fromBuy)
```

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	contracts/MeeKyunDosa.sol#L338,478,514,1572
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

Recommendation

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

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	contracts/MeeKyunDosa.sol#L1050
Status	Unresolved

Description

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

```
uint256 iterations
uint256 lastProcessedIndex
uint256 claims
```

Recommendation

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

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/MeeKyunDosa.sol#L1011,1012,1013,1014,1149,1153,1165,1166,1167
Status	Unresolved

Description

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

```
dividendsFromBuy = address(this).balance.mul(buyAmount).div(totalAmount)
            .mul(buyRewardsFee).div(buyRewardsFee + buyMarketingFees + buyBBPFee)
```

Recommendation

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

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/MeeKyunDosa.sol#L930,931,932,933,934,1028,1050,1140,1141,1145,1146,1485,1531
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.

```
uint256 RewardsFee
uint256 deadFees
uint256 marketingFees
uint256 liquidityFee
uint256 BBPFees
uint256 burntokens
uint256 iterations
uint256 lastProcessedIndex
uint256 claims
uint256 dividendsFromBuy
uint256 dividendsFromSell
uint256 marketingPayout
uint256 devPayout
bool success
```

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	contracts/MeeKyunDosa.sol#L451,1231
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.

```
string memory _name
string memory _symbol
```

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	contracts/MeeKyunDosa.sol#L280,723,1236,1475
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.

```
_owner = msgSender
marketingWallet = payable(wallet)
defaultToken = token
```

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/MeeKyunDosa.sol#L1504
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.

```
MeeKyunDosaContract.transfer(account, received)
```

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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-

	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-

	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-

DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawDividend	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		

SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
DividendPayingToken	Implementation	ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
		External	Payable	-
	distributeDividends	Public	Payable	-
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
MeeKyunDosa	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	decimals	Public		-
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	setPresaleWallet	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner

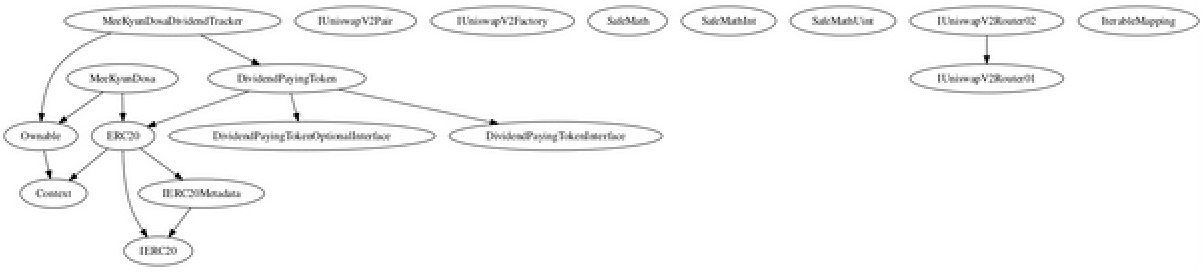
	setExcludeFees	Public	✓	onlyOwner
	setExcludeDividends	Public	✓	onlyOwner
	setIncludeDividends	Public	✓	onlyOwner
	setCanTransferBefore	External	✓	onlyOwner
	setLimitsInEffect	External	✓	onlyOwner
	setGasPriceLimit	External	✓	onlyOwner
	setcooldowntimer	External	✓	onlyOwner
	setmaxWallet	External	✓	onlyOwner
	Sweep	External	✓	onlyOwner
	setmaxTX	External	✓	onlyOwner
	setSwapTriggerAmount	Public	✓	onlyOwner
	enableSwapAndLiquify	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	setAllowCustomTokens	Public	✓	onlyOwner
	setAllowAutoReinvest	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	transferAdmin	Public	✓	onlyOwner
	updateTransferFee	Public	✓	onlyOwner
	updateFees	Public	✓	onlyOwner
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-

	getNumberOfDividendTokenHolders	External		-
	setAutoClaim	External	✓	-
	setReinvest	External	✓	-
	setDividendsPaused	External	✓	onlyOwner
	isExcludedFromAutoClaim	External		-
	isReinvest	External		-
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	updatePayoutToken	Public	✓	onlyOwner
	getPayoutToken	Public		-
	setMinimumTokenBalanceForAutoDividends	Public	✓	onlyOwner
	setMinimumTokenBalanceForDividends	Public	✓	onlyOwner
	addLiquidity	Private	✓	
	forceSwapAndSendDividends	Public	✓	onlyOwner
	swapAndSendDividends	Private	✓	
	airdropToWallets	External	✓	onlyOwner
MeeKyunDosaDividendTracker	Implementation	DividendPayingToken, Ownable		
		Public	✓	DividendPayingToken
	decimals	Public		-
	name	Public		-
	symbol	Public		-
	_transfer	Internal		
	withdrawDividend	Public		-
	isExcludedFromAutoClaim	External		onlyOwner
	isReinvest	External		onlyOwner

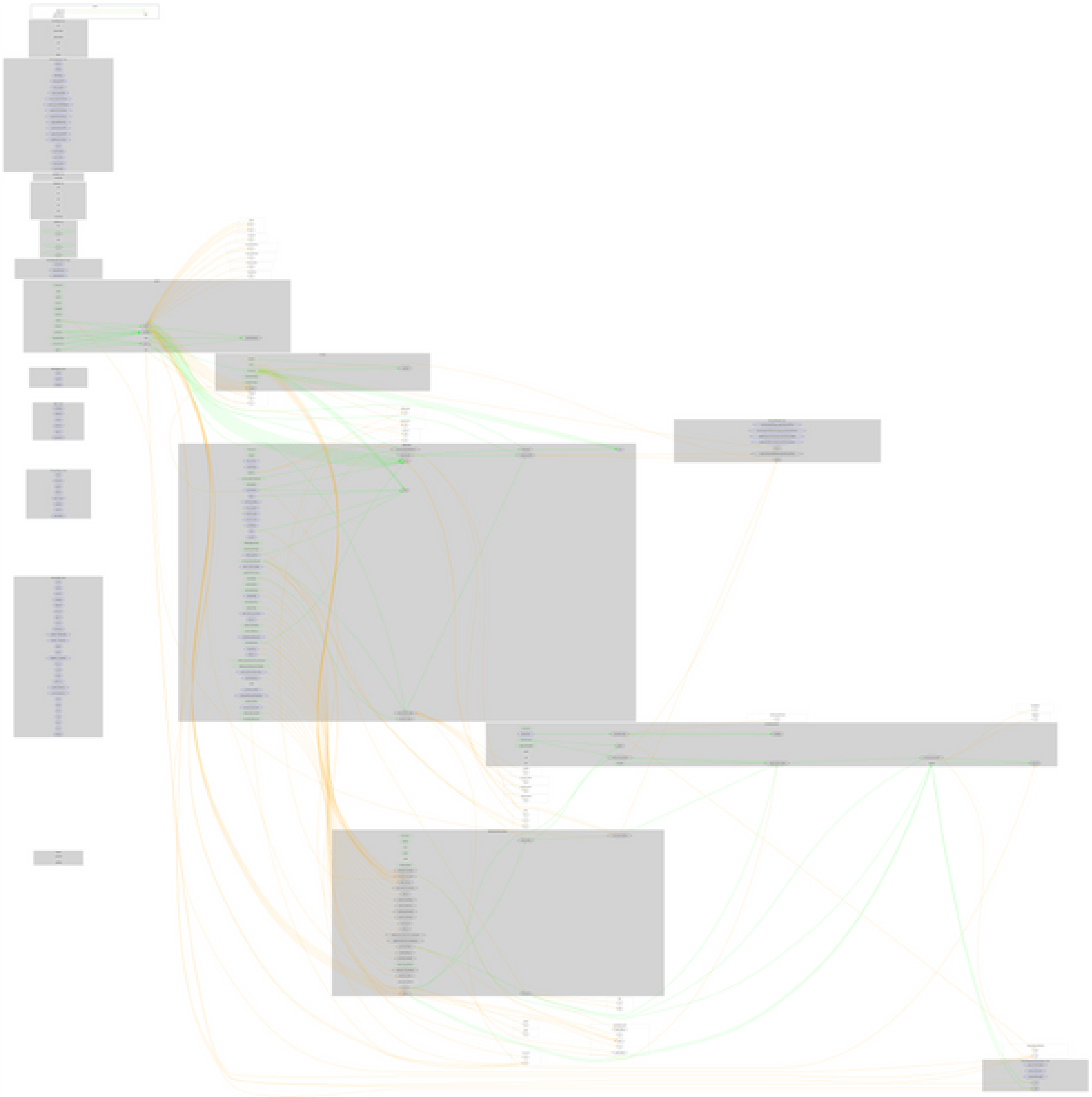
	setAllowCustomTokens	External	✓	onlyOwner
	setAllowAutoReinvest	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	includeFromDividends	External	✓	onlyOwner
	setAutoClaim	External	✓	onlyOwner
	setReinvest	External	✓	onlyOwner
	setMinimumTokenBalanceForAutoDividends	External	✓	onlyOwner
	setMinimumTokenBalanceForDividends	External	✓	onlyOwner
	setDividendsPaused	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
	updateUniswapV2Router	Public	✓	onlyOwner
	updatePayoutToken	Public	✓	onlyOwner
	getPayoutToken	Public		-
	_reinvestDividendOfUser	Private	✓	
	_withdrawDividendOfUser	Internal	✓	
IterableMapping	Library			
	get	Internal		
	getIndexOfKey	Internal		
	getKeyAtIndex	Internal		
	size	Internal		
	set	Internal	✓	

	remove	Internal	✓	

Inheritance Graph



Flow Graph



Summary

MeeKyunDosa contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. MeeKyunDosa is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues. The Contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 10% buy/sell fees and 10% transfer fees.

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