Cyberscope

Audit Report

MemeFamilyHoliday

December 2022

Type	BEP20
Network	BSC
Address	0xff99fca6a9d5e22e836f7a6e1e0e33bfdf3dad90
Audited by	© cyberscope

Table of Contents

Review

Contract Name	MEMEFAMILYHOLIDAY
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	2000 runs
Explorer	https://bscscan.com/address/0xff99fca6a9d5e22e836f7a6e1e0e33bfdf3dad90
Address	0xff99fca6a9d5e22e836f7a6e1e0e33bfdf3dad90
Network	BSC
Symbol	$MFH
Decimals	9
Total Supply	1,000,000,000,000,000

Audit Updates

Initial Audit	24 Dec 2022

Source Files

Filename	SHA256
MEMEFAMILYHOLIDAY.sol	a93ebfbbf242acdb182e775906138de32eb42bf9798a3a24008b5da6185ae017

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
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	CR	Code Repetition	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L1075
Status	Unresolved

Description

The contract sends funds to the `projecktrecievers` and `marketingreciever` as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address is a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(projecktreciever1).transfer(amountBNBforProjekt1);
payable(projecktreciever2).transfer(amountBNBforProjekt2);
payable(projecktreciever3).transfer(amountBNBforProjekt3);
payable(projecktreciever4).transfer(amountBNBforProjekt4);
payable(marketingreciever).transfer(amountBNBforProjekt5);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L1061,1073
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 amountBNBReflection3 =
amountBNB.mul(reflectionFee).div(totalBNBFee).mul(Div3).div(10**2);
uint256 amountBNBReflection4 =
amountBNB.mul(reflectionFee).div(totalBNBFee).mul(Div4).div(10**2);
...
uint256 amountBNBforProjekt4 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(22).div(10**2);
uint256 amountBNBforProjekt5 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(12).div(10**2);
```

Recommendation

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

CR - Code Repetition

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L205,353,500,648
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
contract DividendDistributor is IDividendDistributor {}
contract DividendDistributor1 is IDividendDistributor {}
contract DividendDistributor2 is IDividendDistributor {}
contract DividendDistributor3 is IDividendDistributor {}
```

All the above statements repeat the same code blocks with the different of the token address and name.

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help to reduce the complexity and size of your contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L9
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 on 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, overhead and increases unnecessary the gas consuption.

```
library SafeMath {

}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater that `0.8.0` then the pure Solidity arithmetic operations produces the same result. If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement. Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L216,217,230,363,364,377,510,511,524,658,659,672,799,800,801,802,804,805,806,812,832
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 decreases gas consumption of the corresponding transaction.

```
IBEP20 BUSD = IBEP20(0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
IBEP20 USDT = IBEP20(0x55d398326f99059fF775485246999027B3197955)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
IBEP20 DOGE = IBEP20(0xbA2aE424d960c26247Dd6c32edC70B295c744C43)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
IBEP20 BTC = IBEP20 (0x7130d2A12B9BCbFAe4f2634d864A1Ee1Ce3Ead9c)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
address BUSD = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56
address USDT = 0x55d398326f99059fF775485246999027B3197955

...
```

Recommendation

Constant state variables can be useful when you want 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 advices to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L155,208,216,217,252,252,356,363,364,399,399,503,510,511,546,546,651,658,659,694,694,799,800,801,802,803,804,805,806,810,812,813,814,817,818,832,836,837,838,839,938,1134,1134,1134,1134,1168,1168,1205,1205,1205,1205,1205,1215,1215,1215,1215,1215,1215,1224,1224,1224,1224,1232,1232,1236,1236,1240,1240,1240,1240,1240,1240,1240,1240
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 your Solidity code, making it easier for others to understand and work with.

The followings are 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 your code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
address _token
IBEP20 BUSD = IBEP20(0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 _minDistribution
uint256 _minPeriod
address _token
IBEP20 USDT = IBEP20(0x55d398326f99059fF775485246999027B3197955)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 _minDistribution
uint256 _minPeriod
address _token
IBEP20 DOGE = IBEP20(0xbA2aE424d960c26247Dd6c32edC70B295c744C43)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

You can 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	MEMEFAMILYHOLIDAY.sol#L799,800,801,802,806
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.

```
address BUSD = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56
address USDT = 0x55d398326f99059fF775485246999027B3197955
address DOGE = 0xbA2aE424d960c26247Dd6c32edC70B295c744C43
address BTC = 0x7130d2A12B9BCbFAe4f2634d864A1Ee1Ce3Ead9c
address DEAD_NON_CHECKSUM = 0x000000000000000000000000000000000000dEaD
```

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	MEMEFAMILYHOLIDAY.sol#L253,400,547,695,1136,1144,1160,1165,1206,1225,1234,1237
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.

```
minPeriod = _minPeriod
minPeriod = _minPeriod
minPeriod = _minPeriod
minPeriod = _minPeriod
autoBuybackCap = _cap
buybackMultiplierNumerator = numerator
_maxTxAmount = amount
_maxWallet = amount
liquidityFee = _liquidityFee
Div1 = _DIV1
swapThreshold = _amount
targetLiquidity = _target
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, you can help to ensure that the contract 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	MEMEFAMILYHOLIDAY.sol#L1148
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 launched() internal view returns (bool) {
        return launchedAt != 0;
    }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L928
Status	Unresolved

Description

The boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false. it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(buyBacker[msg.sender] == true, "")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L1058,1059,1060,1061,1069,1070,1071,1072,1073
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 lose of prediction.

```
uint256 amountBNBReflection1 =
amountBNB.mul(reflectionFee).div(totalBNBFee).mul(Div1).div(10**2)
uint256 amountBNBReflection2 =
amountBNB.mul(reflectionFee).div(totalBNBFee).mul(Div2).div(10**2)
uint256 amountBNBReflection3 =
amountBNB.mul(reflectionFee).div(totalBNBFee).mul(Div3).div(10**2)
uint256 amountBNBReflection4 =
amountBNB.mul(reflectionFee).div(totalBNBFee).mul(Div4).div(10**2)
uint256 amountBNBforProjekt1 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(22).div(10**2)
uint256 amountBNBforProjekt2 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(22).div(10**2)
uint256 amountBNBforProjekt3 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(22).div(10**2)
uint256 amountBNBforProjekt4 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(22).div(10**2)
uint256 amountBNBforProjekt5 =
amountBNB.mul(projektFee).div(totalBNBFee).mul(12).div(10**2)
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MEMEFAMILYHOLIDAY.sol#L144,1216,1217,1218,1219,1220,1221
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 = adr
autoLiquidityReceiver = _autoLiquidityReceiver
projecktreciever1 = _projecktreciever1
projecktreciever2 = _projecktreciever2
projecktreciever3 = _projecktreciever3
projecktreciever4 = _projecktreciever4
marketingreciever = _marketingreciever
```

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	MEMEFAMILYHOLIDAY.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 you to specify a minimum version of the Solidity compiler that must be used to compile your contract code. This is useful because it allows you to ensure that your contract will be compiled using a version of the compiler that is known to be compatible with your code.

```
pragma solidity ^0.8.17;
```

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	MEMEFAMILYHOLIDAY.sol#L320,467,615,763
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.

```
BUSD.transfer(shareholder, amount)
USDT.transfer(shareholder, amount)
DOGE.transfer(shareholder, amount)
BTC.transfer(shareholder, 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**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-

	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	authorize	Public	✓	onlyOwner
	unauthorize	Public	✓	onlyOwner
	isOwner	Public		-
	isAuthorized	Public		-
	transferOwnership	Public	✓	onlyOwner
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IDividendDistributor	Interface			
	setDistributionCriteria	External	✓	-
	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-

DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
DividendDistributor1	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	

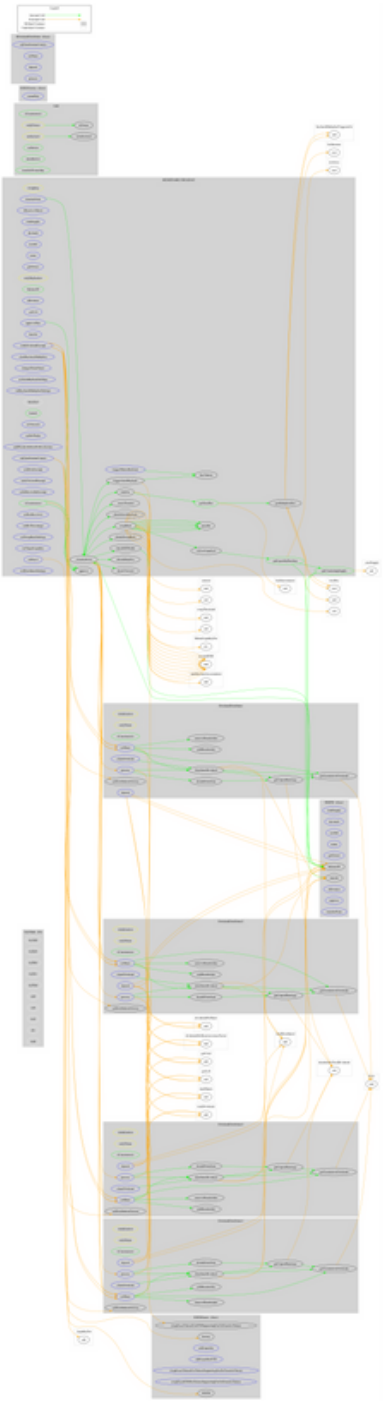
DividendDistributor2	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
DividendDistributor3	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
MEMEFAMILY	Implementation	IBEP20,		

HOLIDAY		Auth		
		Public	✓	Auth
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	_goLive	External	✓	onlyOwner
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	checkMXWallet	Internal		
	checkTxLimit	Internal		
	shouldTakeFee	Internal		
	getTotalFee	Public		-
	getMultipliedFee	Public		-
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	shouldAutoBuyback	Internal		
	triggerMemeBuyback	External	✓	authorized
	clearBuybackMultiplier	External	✓	authorized
	triggerAutoBuyback	Internal	✓	
	buyTokens	Internal	✓	swapping

	changeTokenName	External	✓	authorized
	setAutoBuybackSettings	External	✓	authorized
	setBuybackMultiplierSettings	External	✓	authorized
	launched	Internal		
	launch	Public	✓	authorized
	setTxLimit	External	✓	authorized
	setMxWallet	External	✓	authorized
	addPresaleAddressForExclusions	External	✓	authorized
	setIsDividendExempt	External	✓	authorized
	setIsFeeExempt	External	✓	authorized
	setIsTxLimitExempt	External	✓	authorized
	setIsMaxwalletExempt	External	✓	authorized
	setFees1	External	✓	authorized
	setFeeReceivers	External	✓	authorized
	setDivPercentage	External	✓	authorized
	setSwapBackSettings	External	✓	authorized
	setTargetLiquidity	External	✓	authorized
	setDistributionCriteria	External	✓	authorized
	setDistributorSettings	External	✓	authorized
	getCirculatingSupply	Public		-
	getLiquidityBacking	Public		-
	isOverLiquified	Public		-

Inheritance Graph

Flow Graph



Summary

MemeFamilyHoliday is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error 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 25% 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