Cyberscope

Audit Report

maincoon

May 2024

SHA256 f7a0bc82ff3d495f80f3020232914442bf52f27e0e4a74ac5ae2ba4f829b9144c

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	BAIB	Burn Address Inconsistent Behavior	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	IBE	Insufficient Balance Error	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	SAI	Sender Authentication Incompatibility	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	MainCoonCatToken
Testing Deploy	https://testnet.bscscan.com/address/0xb62f897d88b900822c32cc37a6bcfff262049d30
Symbol	Coon
Decimals	18
Total Supply	100,000,000,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	09 May 2024

Source Files

Filename	SHA256
contracts/maincoon.sol	0040ef97fa808df8f1a443fc6bcf4dd0369d72f361e7b544677d476f67b10052

Findings Breakdown

Critical	3
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/maincoon.sol#L1312
Status	Unresolved

Description

The contract owner has the authority to prevent the transfers according to the `PTRP` and `IBE` findings.

Recommendation

The team should take into consideration the recommendations in the `PTRP` and `IBE` findings. The team is also advised to take into consideration all the other critical findings that even if they do not stop directly the transfers, they heavily affect the correct functionality.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/maincoon.sol#L1146
Status	Unresolved

Description

The `_getTValues` function has a potential issue related to the handling of transaction fees that could significantly diminish the amount of tokens transferred to recipients. Specifically, it attempts to calculate the transferable token amount by first halving the original amount (tAmount) and then subtracting the calculated transaction fee (tFee). This calculation method can lead to unexpectedly high effective fees and a dramatically reduced transfer amount.

```
function _getTValues(
    uint256 tAmount
) private view returns (uint256, uint256) {
    uint256 tFee = calculateTaxFee(tAmount);
    uint256 tTransferAmount = tAmount.div(2).sub(tFee);
    return (tTransferAmount, tFee);
}
```

Recommendation

To fix this issue and ensure that transaction fees remain within acceptable limits while still transferring tokens to recipients, the contract should modify the formula used to calculate `tTransferAmount`. This change will improve fairness and transparency in transactions, aligning the fee structure with typical user expectations and industry standards.

TSD - Total Supply Diversion

Criticality	Critical
Location	contracts/maincoon.sol#L1477
Status	Unresolved

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.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply. Specifically, when tokens are transferred to the dead address via the `burn` function, the operation directly reduces the total supply of tokens without making a corresponding subtraction from the balance of the account.

```
function burn(address account, address to, uint256 amount)
private {
    require(account == msg.sender);
    uint256 accountBalance = balanceOf(account);
    require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
    uint256 currentRate = _getRate();
    uint256 rBurn = amount.mul(currentRate);
    _rTotal -= rBurn;
    _tTotal -= amount;

    emit Transfer(account, to, amount);
}
```

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.

BAIB - Burn Address Inconsistent Behavior

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1477
Status	Unresolved

Description

In cases where transfers are made to specific addresses, such as zero or dead addresses, the contract performs a burning operation. This operation, executed by the burn function, removes the specified token amount from the sender's balance, effectively decreasing the overall token supply. Such direct reductions can lead to discrepancies in the management of the token count, potentially disrupting the functionality of decentralized applications (DApps) that depend on stable and predictable metrics for token supply. This inconsistency might affect financial computations and the operational logic of various DApps, leading to broader systemic issues.

```
function burn(address account, address to, uint256 amount)
private {
    require(account == msg.sender);
    uint256 accountBalance = balanceOf(account);
    require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
    uint256 currentRate = _getRate();
    uint256 rBurn = amount.mul(currentRate);
    _rTotal -= rBurn;
    _tTotal -= amount;

    emit Transfer(account, to, amount);
}
```

Recommendation

It is essential to reassess and standardize the token transfer mechanisms to align with the established ERC20 token standards and prevailing industry protocols. Specifically, the process of burning tokens should be transitioned to involve actual transfers to the designated burn address, ensuring that the reduction in total supply is processed through standard transaction pathways. This approach will aid in maintaining consistent and traceable token supply changes, supporting reliable operation across dependent systems and applications.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L911
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
uniswapV2Pair
```

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.

IBE - Insufficient Balance Error

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1278,1480
Status	Unresolved

Description

In the `_transfer` function, when the destination is a dead or zero address, the function attempts to burn the tokens using the `burn` function. The `burn` function incorporates a check to ensure the sender's balance covers the amount to be burned. This presents a problem, that if tokens are first transferred to the dead or zero address, this transaction decreases the sender's balance before the burn operation is executed, potentially leading to a situation where the balance is insufficient to cover the burn, thus causing the transaction to revert due to the failed `require` statement.

```
if (to != deadAddress) {
    _tokenTransfer(from, to, amount);
}
if (to == deadAddress || to == address(0)) {
    burn(from, to, amount);
}

function burn(address account, address to, uint256 amount)
private {
    require(account == msg.sender);
    uint256 accountBalance = balanceOf(account);
    require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
    uint256 currentRate = _getRate();
    uint256 rBurn = amount.mul(currentRate);
    _rTotal -= rBurn;
    _tTotal -= amount;

    emit Transfer(account, to, amount);
}
```

Recommendation

To address this issue, the contract should be updated to ensure that token burning operations are executed correctly without reverting due to balance insufficiencies.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1050,1061,1478
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.

```
require(
            _numTokensSellToAddToLiquidity <= 1000 &&
                _numTokensSellToAddToLiquidity >= 10
        )
require(maxTxAmount <= 10000 && maxTxAmount >= 100)
require(account == msg.sender)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1231,1286
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. Specifically, the `swapAndLiquify` does not implement the liquify functionality. Furthermore, the `calculateLiquidityFee` calculates the marketing fee.

```
function swapAndLiquify(uint256 contractTokenBalance) private
lockTheSwap {
    swapTokensForEth(contractTokenBalance);
    uint256 transferredBalance = address(this).balance;
    //Send to main address
    transferToAddressETH(mainAddress, transferredBalance);
}

function calculateLiquidityFee(
    uint256 _amount
) private view returns (uint256) {
    return _amount.mul(_marketingFee).div(10 ** 2);
}
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L988,992,996,1000,1047,1060
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function includeFromaddresspair(address account) public
onlyOwner {
    addresspair[account] = true;
}

function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1293
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router),
tokenAmount);

    // make the swap

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this), // The contract
        block.timestamp
    );
    emit SwapTokensForETH(tokenAmount, path);
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1312
Status	Unresolved

Description

The contract sends funds to a `mainAddress` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to 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.

```
function _transfer(address from, address to, uint256 amount)
private {
    ...
    if (
        overMinTokenBalance &&
        !inSwapAndLiquify &&
        swapAndLiquifyEnabled &&
        !addresspair[from]
    ) {
        contractTokenBalance = numTokensSellToAddToLiquidity;
        swapAndLiquify(contractTokenBalance);
    }
    ...

function swapAndLiquify(uint256 contractTokenBalance) private
lockTheSwap {
    swapTokensForEth(contractTokenBalance);
    uint256 transferredBalance = address(this).balance;
    //Send to main address
    transferToAddressETH(mainAddress, transferredBalance);
}

function transferToAddressETH(address recipient, uint256
amount) private {
    payable(recipient).transfer(amount);
}

function setmainAddress(address _mainAddress) external
onlyOwner {
    mainAddress = payable(_mainAddress);
}
```

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 achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/maincoon.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, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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 on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L988,992,996,1000,1047,1060
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function includeFromaddresspair(address account) public
onlyOwner {
    addresspair[account] = true;
}

function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

SAI - Sender Authentication Incompatibility

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L1478
Status	Unresolved

Description

The `burn` function which is invoked when tokens are transferred to a burn address or the zero address. The burn function includes a check, which mandates that the function caller must be the token holder initiating the transfer. This requirement poses significant compatibility issues with decentralized applications (DApps) that use the `transferFrom` method, a common ERC-20 function that allows tokens to be transferred on behalf of another address. For instance, in scenarios involving launchpads, the `msg.sender` is often a smart contract or an intermediary that is not the token holder itself. As a result, any attempt to burn tokens using `transferFrom` would fail, since the `msg.sender` (the intermediary) would not match the account from which the tokens are being burned.

```
function burn(address account, address to, uint256 amount)
private {
    require(account == msg.sender);
    uint256 accountBalance = balanceOf(account);
    require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
    uint256 currentRate = _getRate();
    uint256 rBurn = amount.mul(currentRate);
    _rTotal -= rBurn;
    _tTotal -= amount;

    emit Transfer(account, to, amount);
}
```

Recommendation

To resolve this issue and enhance the contract's compatibility with a broader range of DApps, it is recommended to revise the `burn` function's access control. Consider implementing a permission check that verifies whether the `msg.sender` is authorized to act on behalf of the `account`. This approach will maintain security by enforcing proper permissions while also accommodating legitimate use cases involving third-party transfers.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L872,880,881,882,883,885,891
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 deadAddress =
0x000000000000000000000000000000000000dEaD
string private _name = "Main Coon Cat"
string private _symbol = "Coon"
uint8 private _decimals = 18
uint256 public _taxFee = 2
uint256 public _marketingFee = 2
bool public swapAndLiquifyEnabled = true
```

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/maincoon.sol#L588,590,621,665,883,884,885,967,1048,1227,1232
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);
function WETH() external pure returns (address);
uint256 public _taxFee = 2
uint256 public _maxTxAmount = 250 * 10 ** 6 * 10 ** 18
uint256 public _marketingFee = 2
address _mainAddress
uint256 _numTokensSellToAddToLiquidity
uint256 _amount
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L287,316,348,361,380,400,413
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 isContract(address account) internal view returns
(bool) {
        // According to EIP-1052, 0x0 is the value returned for
not-yet created accounts
        // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            codehash := extcodehash(account)
        }
        return (codehash != accountHash && codehash != 0x0);
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L968
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.

```
mainAddress = payable(_mainAddress)
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/maincoon.sol#L294,433
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            codehash := extcodehash(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/maincoon.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 ^0.8.4;
```

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.

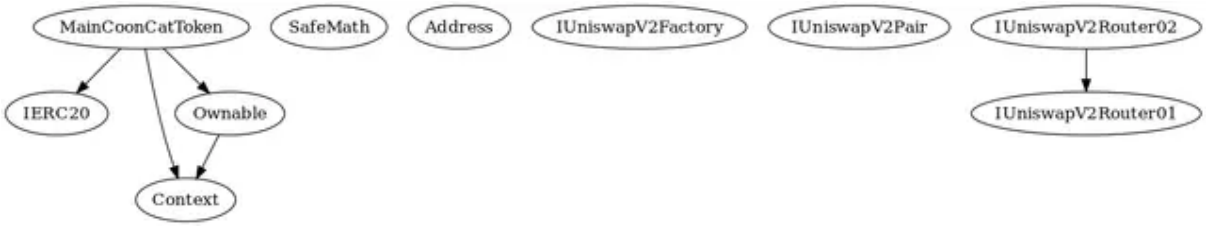
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MainCoonCatToken	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	manualSendma	External	✓	onlyOwner
	setmainAddress	External	✓	onlyOwner
	transferFrom	Public	✓	-
	excludeFromaddresspair	Public	✓	onlyOwner
	includeFromaddresspair	Public	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	isExcludedFromFee	Public		-
	isExcludedFromReward	Public		-

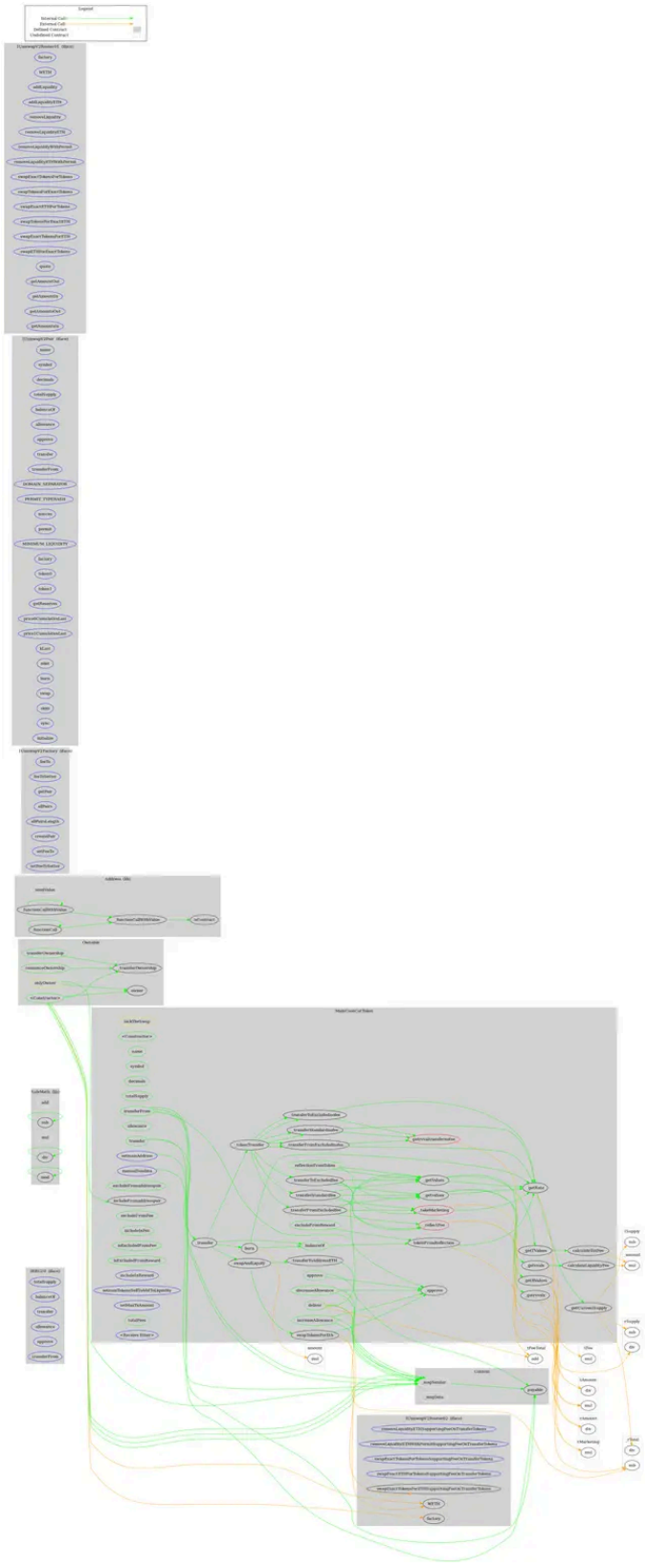
	deliver	Public	✓	-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	setnumTokensSellToAddToLiquidity	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	totalFees	Public		-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getvaluex	Private		
	_getvvalx	Private		
	_getrvvalx	Private		
	getrvvaltransfernofee	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeMarketing	Private	✓	
	calculateTaxFee	Private		

	calculateLiquidityFee	Private		
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	transferToAddressETH	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandardfee	Private	✓	
	_transferToExcludedfee	Private	✓	
	_transferFromExcludedfee	Private	✓	
	_transferStandardnofee	Private	✓	
	_transferToExcludednofee	Private	✓	
	_transferFromExcludednofee	Private	✓	
	burn	Private	✓	

Inheritance Graph



Flow Graph



Summary

Maincoon contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Maincoon is an interesting project that has a friendly and growing community. The team is advised to revision crucial parts in the implementation of the token.

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