Cyberscope

Audit Report

maincoon

May 2024

SHA256 2fdc35045611ca2e75f6cf429bed71232496f2f3b79272113c869bdc131c8b2a

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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IIO	Incorrect Initialization Order	Unresolved
●	TSD	Total Supply Diversion	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	BAIB	Burn Address Inconsistent Behavior	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	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

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 4
Review 6
Audit Updates 6
Source Files 6
Overview 7
Findings Breakdown 8
ST - Stops Transactions 9
Description 9
Recommendation 9
IIO - Incorrect Initialization Order 10
Description 10
Recommendation 10
TSD - Total Supply Diversion 11
Description 11
Recommendation 12
AOI - Arithmetic Operations Inconsistency 13
Description 13
Recommendation 13
BAIB - Burn Address Inconsistent Behavior 14
Description 14
Recommendation 15
MEE - Missing Events Emission 16
Description 16
Recommendation 16
PAV - Pair Address Validation 17
Description 17
Recommendation 17
PLPI - Potential Liquidity Provision Inadequacy 18
Description 18
Recommendation 19
PTRP - Potential Transfer Revert Propagation 20
Description 20
Recommendation 21
RSML - Redundant SafeMath Library 22
Description 22
Recommendation 22
RSW - Redundant Storage Writes 23

Description 23
Recommendation 23
L02 - State Variables could be Declared Constant 24
Description 24
Recommendation 24
L04 - Conformance to Solidity Naming Conventions 25
Description 25
Recommendation 26
L09 - Dead Code Elimination 27
Description 27
Recommendation 28
L16 - Validate Variable Setters 29
Description 29
Recommendation 29
L17 - Usage of Solidity Assembly 30
Description 30
Recommendation 30
Functions Analysis **31**
Inheritance Graph **39**
Flow Graph **40**
Summary **41**
Disclaimer **42**
About Cyberscope **43**

Review

Contract Name	MainCoonCatToken
Symbol	Coon
Decimals	18
Total Supply	100,000,000,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	09 May 2024 https://github.com/cyberscope-io/audits/blob/main/maincoon/v1/audit.pdf
Corrected Phase 2	14 May 2024

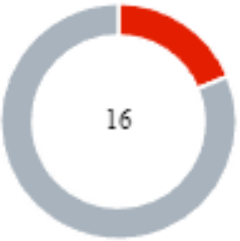
Source Files

Filename	SHA256
maincooncopy.sol	2fdc35045611ca2e75f6cf429bed71232496f2f3b79272113c869bdc131c8b2a

Overview

The smart contract cannot be deployed with the default configuration as described in the `IIO` finding.

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	13	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 `PAV` findings.

Recommendation

The team should take into consideration the recommendations in the `PTRP` and `PAV` 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.

IIO - Incorrect Initialization Order

Criticality	Critical
Location	maincooncopy.sol#L934
Status	Unresolved

Description

An issue is present within the constructor related to the initialization order of the Uniswap V2 Router and its associated WETH address. Specifically, the line `pairToSwap = uniswapV2Router.WETH()` is executed before the `uniswapV2Router` is initialized. At this point in the constructor, `uniswapV2Router` is still uninitialized and holds the default value, which is the zero address. Consequently, invoking `WETH()` on an uninitialized address results in `pairToSwap` being set to an unintended value, causing the deployment to fail.

```
pairToSwap = uniswapV2Router.WETH();
address _uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
    .createPair(address(this), _uniswapV2Router.WETH());
uniswapV2Router = _uniswapV2Router;
```

Recommendation

To resolve this issue and ensure the correct initialization of the `pairToSwap` variable, the `uniswapV2Router` should be properly initialized before invoking its `WETH()` function. Specifically, use the `_uniswapV2Router`, which is set correctly above, instead of the uninitialized `uniswapV2Router`. This change ensures that `pairToSwap` correctly references the `WETH` address associated with the Uniswap V2 Router, thereby maintaining the intended functionality of the token contract and allowing it to be deployed successfully.

TSD - Total Supply Diversion

Criticality	Critical
Location	contracts/maincooncopy.sol#1590
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 {
    uint256 currentRate = _getRate();
    uint256 rBurn = amount.mul(currentRate);
    if (_isExcluded[account]) {
        _tOwned[account] = _tOwned[account] - amount;
    }
    _rOwned[account] = _rOwned[account].sub(rBurn);
    _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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	maincooncopy.sol#L1600,1601
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
_rOwned[account] = _rOwned[account].sub(rBurn);
_rTotal -= rBurn;
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

BAIB - Burn Address Inconsistent Behavior

Criticality	Minor / Informative
Location	contracts/maincooncopy.sol#L1590
Status	Unresolved

Description

In cases where transfers are made to specific addresses, such as 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 {
    uint256 currentRate = _getRate();
    uint256 rBurn = amount.mul(currentRate);
    if (_isExcluded[account]) {
        _tOwned[account] = _tOwned[account] - amount;
    }
    _rOwned[account] = _rOwned[account].sub(rBurn);
    _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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/maincooncopy.sol#L1094,1108,1116
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 setMaxTxAmount(uint256 maxTxAmount) external onlyOwner
{
    require(
        maxTxAmount <= 10000 && maxTxAmount >= 100,
        "Amount exceeds the possible maxTxAmount."
    );
    _maxTxAmount = maxTxAmount * 10**6 * 10**18;
}

function setPairToSwap(address pair) external onlyOwner {
    pairToSwap = pair;
}
...
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	maincooncopy.sol#L1116
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function setPairToSwap(address pair) external onlyOwner {
    pairToSwap = pair;
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	contracts/maincooncopy.sol#L1376
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] = pairToSwap;

    _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/maincooncopy.sol#L1395
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 {
    ...
    ) {
        contractTokenBalance = numTokensSellToAddToMarketing;
        swapAndMarketing(contractTokenBalance);
    }
    ...
}

function swapAndMarketing(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 {
    if (amount > 0) {
        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	maincooncopy.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/maincooncopy.sol#L1094,1108,1116
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 setMaxTxAmount(uint256 maxTxAmount) external onlyOwner
{
    require(
        maxTxAmount <= 10000 && maxTxAmount >= 100,
        "Amount exceeds the possible maxTxAmount."
    );
    _maxTxAmount = maxTxAmount * 10**6 * 10**18;
}

function setPairToSwap(address pair) external onlyOwner {
    pairToSwap = pair;
}
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	maincooncopy.sol#L884,892,893,894,904
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
bool public swapAndMarketingEnabled = 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	maincooncopy.sol#L586,588,619,667,895,896,897,1006,1095,1298,1302
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 _numTokensSellToAddToMarketing
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	maincooncopy.sol#L286,315,347,360,379,399,412
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	maincooncopy.sol#L1007,1117
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)
pairToSwap = pair
```

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	maincooncopy.sol#L293,432
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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-

	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
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	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-

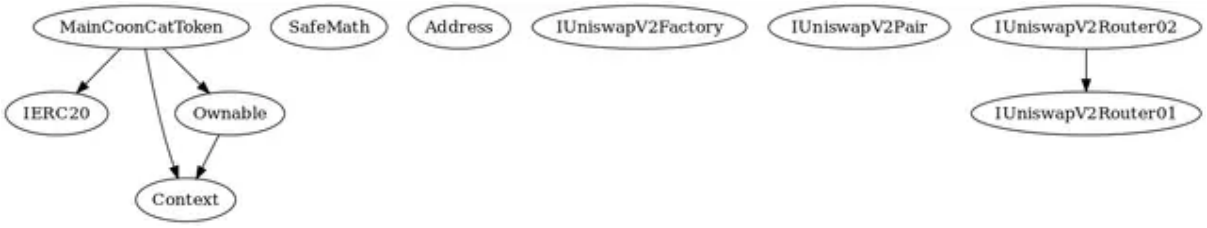
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
MainCoonCatToken	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	setFees	External	✓	onlyOwner
	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
	setNumTokensSellToAddToMarketing	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	setPairToSwap	External	✓	onlyOwner
	increaseAllowance	Public	✓	-

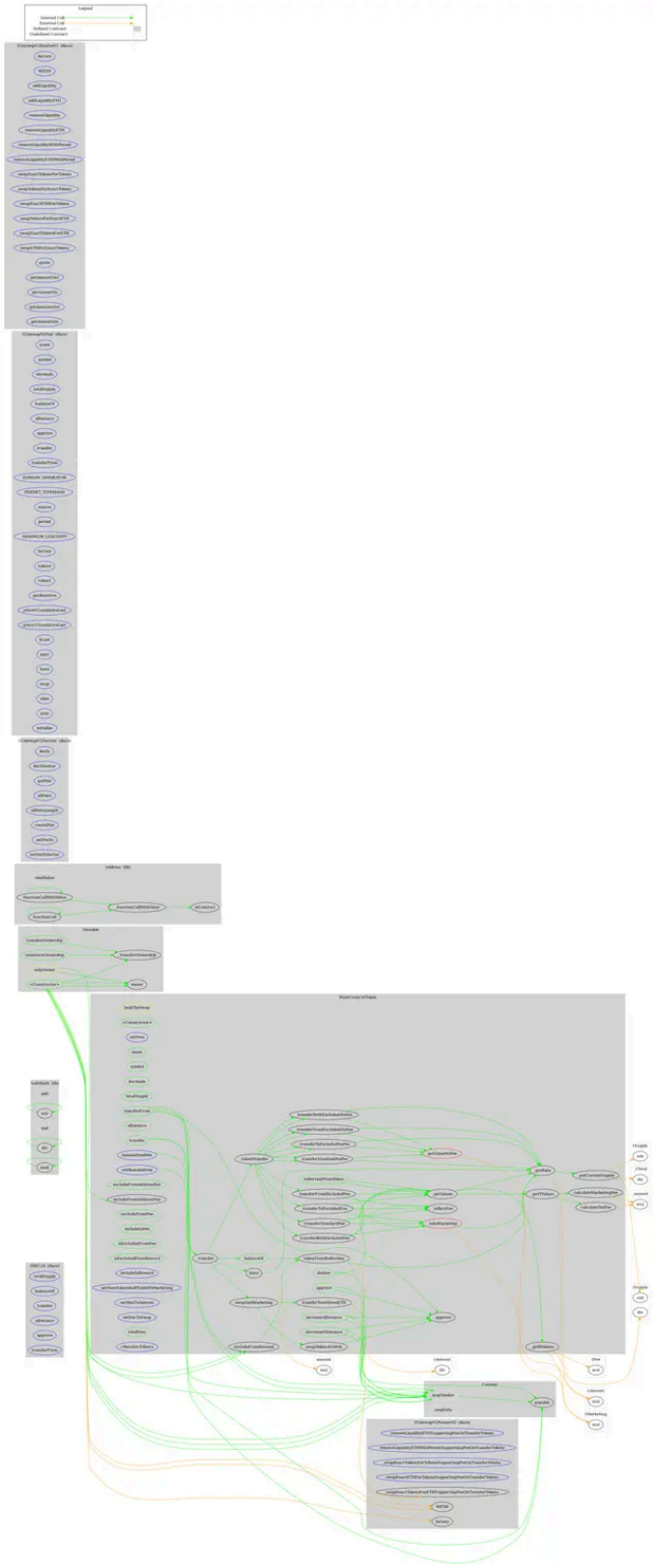
	decreaseAllowance	Public	✓	-
	totalFees	Public		-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	getValueNoFee	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeMarketing	Private	✓	
	calculateTaxFee	Private		
	calculateMarketingFee	Private		
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndMarketing	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	transferToAddressETH	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandardFee	Private	✓	
	_transferToExcludedFee	Private	✓	

	_transferFromExcludedFee	Private	✓	
	_transferBothExcludedFee	Private	✓	
	_transferStandardNoFee	Private	✓	
	_transferToExcludedNoFee	Private	✓	
	_transferFromExcludedNoFee	Private	✓	
	_transferBothExcludedNoFee	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. The team should carefully take into consideration the information given in the Overview section. Furthermore, there are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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