Cyberscope

Audit Report

MEOO WORLD ORDER

May 2024

Network	MATIC
Address	0xdfB50B616c6797C79cA833868cD028d05a6f4D21
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
●	ZD	Zero Division	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	FRV	Fee Restoration Vulnerability	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	CoinManufactory
Compiler Version	v0.8.15+commit.e14f2714
Optimization	No
Explorer	https://polygonscan.com/token/0xdfb50b616c6797c79ca833868cd028d05a6f4d21
Address	0xdfB50B616c6797C79cA833868cD028d05a6f4D21
Network	MATIC
Symbol	MWO
Decimals	18
Total Supply	666000000000
Badge Eligibility	Must Fix Criticals

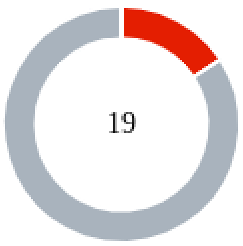
Audit Updates

Initial Audit	02 May 2024

Source Files

Filename	SHA256
CoinManufactory.sol	1e5cbd8819014940350978a9c77d30e2836c73b73b439c82191d704267d11a37

Findings Breakdown



●	Critical	3
●	Medium	0
●	Minor / Informative	16

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	3	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	16	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CoinManufactory.sol#L1402
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` to zero. As a result, the contract may operate as a honeypot.

```
if (sender != owner() && recipient != owner()) {
    require(
        amount <= _maxTxAmount,
        "Transfer amount exceeds the maxTxAmount."
    );
}
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in sections `PTRP` and `ZD`. As a result, the contract might operate as a honeypot.

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.

- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	CoinManufactory.sol#L1330,1334,1339,1344
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the functions with a high percentage value.

```
   function setBurnFee(uint256 burnFee_) external onlyOwner {
        _burnFee = burnFee_;
    }

    function setMarketingFee(uint256 marketingFee_) external
onlyOwner {
        _marketingFee = marketingFee_;
        _combinedLiquidityFee = marketingFee_ + _developerFee +
_charityFee;
    }

    function setDeveloperFee(uint256 developerFee_) external
onlyOwner {
        _developerFee = developerFee_;
        _combinedLiquidityFee = _marketingFee + developerFee_ +
_charityFee;
    }

    function setCharityFee(uint256 charityFee_) external
onlyOwner {
        _charityFee = charityFee_;
        _combinedLiquidityFee = _marketingFee + _developerFee +
charityFee_;
    }
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ZD - Zero Division

Criticality	Critical
Location	CoinManufactory.sol#L1503
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

Specifically the `_combinedLiquidityFee` varaible can be set to zero.

```
transferToAddressETH(
    marketingAddress,
    ((transferredBalance) * _marketingFee) / _combinedLiquidityFee
);
transferToAddressETH(
    developerAddress,
    ((transferredBalance) * _developerFee) / _combinedLiquidityFee
);
transferToAddressETH(
    charityAddress,
    ((transferredBalance) * _charityFee) / _combinedLiquidityFee
);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1503
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.

```
transferToAddressETH(
    marketingAddress,
    ((transferredBalance) * _marketingFee) /
_combinedLiquidityFee
);
transferToAddressETH(
    developerAddress,
    ((transferredBalance) * _developerFee) /
_combinedLiquidityFee
);
transferToAddressETH(
    charityAddress,
    ((transferredBalance) * _charityFee) /
_combinedLiquidityFee
);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1435,1571,1587
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
    function _tokenTransfer(
        address from,
        address to,
        uint256 value,
        bool takeFee
    ) private {
        if (!takeFee) {
            removeAllFee();
        }

        _transferStandard(from, to, value);

        if (!takeFee) {
            restoreAllFee();
        }
    }

    function removeAllFee() private {
        if (_burnFee == 0 && _combinedLiquidityFee == 0)
return;

        _previousBurnFee = _burnFee;
        _previousCombinedLiquidityFee = _combinedLiquidityFee;
        _previousMarketingFee = _marketingFee;
        _previousDeveloperFee = _developerFee;
        _previousCharityFee = _charityFee;

        _burnFee = 0;
        _combinedLiquidityFee = 0;
        _marketingFee = 0;
        _developerFee = 0;
        _charityFee = 0;
    }

    function restoreAllFee() private {
        _burnFee = _previousBurnFee;
        _combinedLiquidityFee = _previousCombinedLiquidityFee;
        _marketingFee = _previousMarketingFee;
        _developerFee = _previousDeveloperFee;
        _charityFee = _previousCharityFee;
    }
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1270
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.

```
_decimals
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1381,1385,1595
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 excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}

function presale(bool _presale) external onlyOwner {
    if (_presale) {
        setSwapAndLiquifyEnabled(false);
        removeAllFee();
        _previousMaxTxAmount = _maxTxAmount;
        _maxTxAmount = totalSupply();
    } else {
        setSwapAndLiquifyEnabled(true);
        restoreAllFee();
        _maxTxAmount = _previousMaxTxAmount;
    }
}
```

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	CoinManufactory.sol#L1523
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.

```
_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
```

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	CoinManufactory.sol#L1611
Status	Unresolved

Description

The contract sends funds to a `recipient` 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.

```
recipient.transfer(amount);
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1411
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
uint256 contractTokenBalance = balanceOf(address(this));
bool overMinimumTokenBalance = contractTokenBalance >=
    minimumTokensBeforeSwap;

if (
    !inSwapAndLiquify &&
    swapAndLiquifyEnabled &&
    recipient == uniswapV2Pair
) {
    if (overMinimumTokenBalance) {
        contractTokenBalance = minimumTokensBeforeSwap;
        swapTokens(contractTokenBalance);
    }
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1244
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event RewardLiquidityProviders(uint256 tokenAmount);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1381,1385
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 excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CoinManufactory.sol#L491,495,923,925,955,1001,1219,1225,1228,1231,1234,1349,1356,1360,1364,1368,1595
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.

```
mapping(address => uint256) internal _balances
uint256 internal _totalSupply
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 _burnFee
uint256 public _marketingFee
uint256 public _developerFee
uint256 public _charityFee
uint256 public _maxTxAmount
uint256 _minimumTokensBeforeSwap
address _marketingAddress
address _developerAddress

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1331,1335,1340,1345,1353,1374
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.

```
_burnFee = burnFee_
_marketingFee = marketingFee_
_developerFee = developerFee_
_charityFee = charityFee_
minimumTokensBeforeSwap = _minimumTokensBeforeSwap
_maxTxAmount = maxTxAmount
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CoinManufactory.sol#L144,168,199,212,231,251,275,294,311,329,346,716,773,1537,1556
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) {
        // This method relies on
extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is
only stored at the end
        // of the constructor execution.

        return account.code.length > 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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CoinManufactory.sol#L1292
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 loss of prediction.

```
minimumTokensBeforeSwap = ((totalSupply_ * 10**decimals_) /
10000) * 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	CoinManufactory.sol#L1357,1361,1365
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.

```
marketingAddress = payable(_marketingAddress)
developerAddress = payable(_developerAddress)
charityAddress = payable(_charityAddress)
```

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	CoinManufactory.sol#L358
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 {
                        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	CoinManufactory.sol#L3
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 ^0.8.0;
```

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	

	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-

	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	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		-

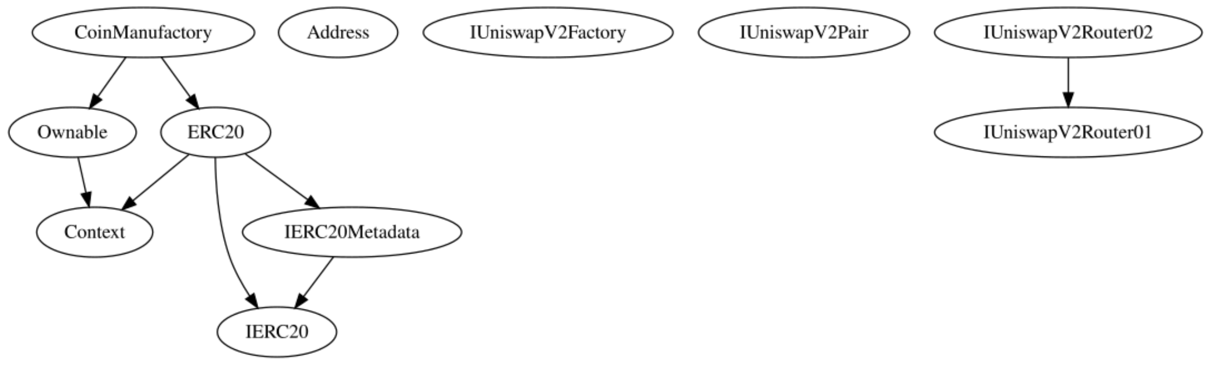
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	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	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
CoinManufactory	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	getBalance	Private		
	decimals	Public		-
	minimumTokensBeforeSwapAmount	Public		-
	setBurnFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	setDeveloperFee	External	✓	onlyOwner

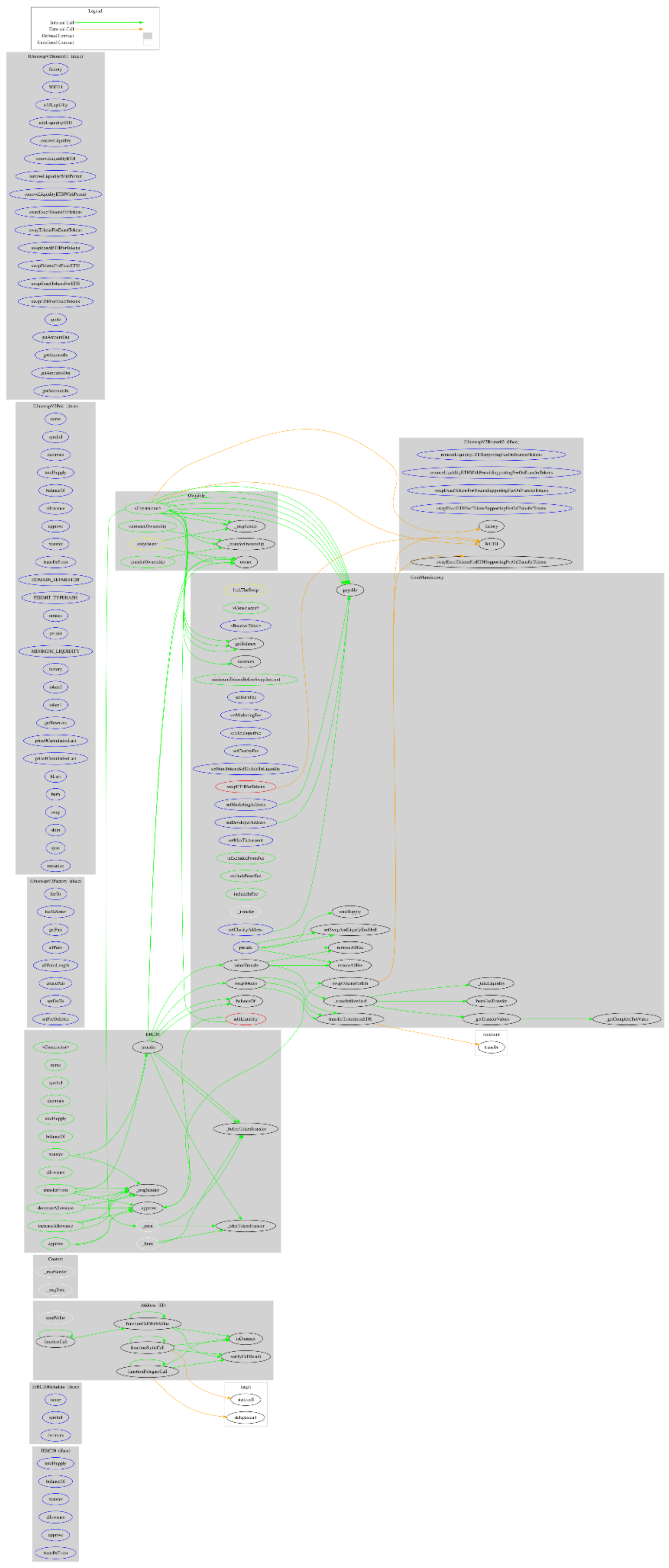
	setCharityFee	External	✓	onlyOwner
	setNumTokensSellToAddToLiquidity	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setDeveloperAddress	External	✓	onlyOwner
	setCharityAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_getTransferValues	Private		
	_getCompleteTaxValue	Private		
	burnFeeTransfer	Private	✓	
	_takeLiquidity	Private	✓	
	swapTokens	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	swapETHForTokens	Private	✓	
	addLiquidity	Private	✓	
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	

	presale	External	✓	onlyOwner
	transferToAddressETH	Private	✓	

Inheritance Graph



Flow Graph



Summary

MEOO WORLD ORDER contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate the fees. 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