Cyberscope

Audit Report

Coffy DeFi

September 2024

Repository https://github.com/CoffyDeFi

Commit 38055f6b7c8ed0ecc84bc0cfee81a9bcedae36a0

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	CCR	Contract Centralization Risk	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	L02	State Variables could be Declared Constant	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

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/CoffyDeFi/coffi/tree/main
Commit	38055f6b7c8ed0ecc84bc0cfee81a9bcedae36a0

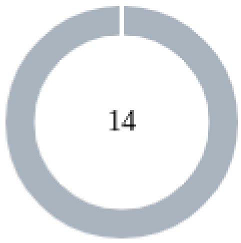
Audit Updates

Initial Audit	19 Sep 2024 https://github.com/cyberscope-io/audits/blob/main/coffi/v1/audit.pdf
Corrected Phase 2	25 Sep 2024
Testing Deploy	https://sepolia.etherscan.io/address/0x74e95e4D4A567eD8Bc7785162032493e071e90df

Source Files

Filename	SHA256
CoffyToken.sol	a9fbc0aa1a97d5bf76baf58d0d6db6854139208cab801417e800186324f5e9a2

Findings Breakdown



- Critical 0
- Medium 0
- Minor / Informative 14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	CoffyDeFi.sol#L879,889,899,910,921,932,939,946,951,965,969,1029,1042,1331,1376,1383
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function setBurnFee(uint256 burnFee_) external onlyOwner {}

function setReflectionFee(uint256 reflectionFee_) public
onlyOwner {}

function setLiquidityPoolFee(uint256 liquidityPoolFee_)
external onlyOwner {}

function setMarketingFee(uint256 marketingFee_) external
onlyOwner {}

function setDeveloperFee(uint256 developerFee_) external
onlyOwner {}

function setMarketingAddress(address _marketingAddress)
external onlyOwner {}

function setDeveloperAddress(address _developerAddress)
external onlyOwner {}

function setSwapAndLiquifyEnabled(bool _enabled) public
onlyOwner {}

function setMaxTxAmount(uint256 maxTxAmount) external onlyOwner
{}

function excludeFromFee(address account) public onlyOwner {}

function includeInFee(address account) public onlyOwner {}

function excludeAccountFromReward(address account) public
onlyOwner {}

function includeAccountinReward(address account) public
onlyOwner {}

function presale(bool _presale) external onlyOwner {}

function recoverETHfromContract() external onlyOwner {}

function recoverTokensfromContract() external onlyOwner {}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	CoffyDeFi.sol#L1248
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 * 10)) /
(_combinedLiquidityFee * 10 - ((_liquidityPoolFee * 10) / 2)));

transferToAddressETH(
developerAddress,
((transferredBalance) * (_developerFee * 10)) /
(_combinedLiquidityFee * 10 - ((_liquidityPoolFee * 10) / 2)));
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CoffyDeFi.sol#L767
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	CoffyDeFi.sol#L965
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;
}
```

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	CoffyDeFi.sol#L1275
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 {
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = uniswapV2Router.WETH();

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

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    tokenAmount,
    0, // accept any amount of ETH
    path,
    address(this),
    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	CoffyDeFi.sol#L1351
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.

```
function transferToAddressETH(
    address payable recipient,
    uint256 amount
    ) private {
    (bool success, ) = recipient.call{value: amount}("");
    if (!success) {
    emit TransferFailed(recipient, amount);
    }
}
```

```
function setMarketingAddress(address _marketingAddress)
external onlyOwner {
    require(_marketingAddress != address(0), "Marketing address
cannot be the zero address");
    address oldAddress = marketingAddress;
    marketingAddress = payable(_marketingAddress);
    emit MarketingAddressUpdated(oldAddress,
_marketingAddress);
}

function setDeveloperAddress(address _developerAddress)
external onlyOwner {
    require(_developerAddress != address(0), "Developer address
cannot be the zero address");
    address oldAddress = developerAddress;
    developerAddress = payable(_developerAddress);
    emit DeveloperAddressUpdated(oldAddress,
_developerAddress);
}
```

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.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	CoffyDeFi.sol#L1293
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
private {
_approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.addLiquidityETH{value: ethAmount}(
    address(this),
    tokenAmount,
    0,
    0,
    owner(),
    block.timestamp
);
}
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	CoffyToken.sol#L801
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.

```
uint256 private minimumTokensBeforeSwap = 100000 ether
```

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	CoffyToken.sol#L492,494,524,564,767,781,784,787,790,793,799,932,939,946,1331,1384,1385
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);
uint8 private constant _decimals = 18
uint256 public _burnFee = 1
uint256 public _reflectionFee = 1
uint256 public _liquidityPoolFee = 1
uint256 public _marketingFee = 1
uint256 public _developerFee = 1
uint256 public _maxTxAmount = 5000000 ether
address _marketingAddress
address _developerAddress
bool _enabled

...
```

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/stable/style-guide.html#naming-conventions.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	CoffyDeFi.sol#L1391,1395,1404,1413,1422,1445,1454,1478
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_
_liquidityPoolFee = liquidityPoolFee_
_marketingFee = marketingFee_
_developerFee = developerFee_
_charityFee = charityFee_
minimumTokensBeforeSwap = _minimumTokensBeforeSwap
_maxTxAmount = maxTxAmount
_reflectionFee = newReflectionFee
```

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	CoffyToken.sol#L355,376,384,858
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 _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual {
        require(sender != address(0),
        "ERC20: transfer from the zero address");
        ...
        unchecked {
            _balances[sender] = senderBalance - amount;
        }
        _balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);
    }

...
```

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	CoffyToken.sol#L1173,1174,1185,1187,1201,1203
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.

```
uint256 tFee = (tAmount * fee) / 100
uint256 rFee = tFee * currentRate
```

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	CoffyDeFi.sol#L1431,1435,1439
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	CoffyToken.sol#L187
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.

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	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
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	✓	-

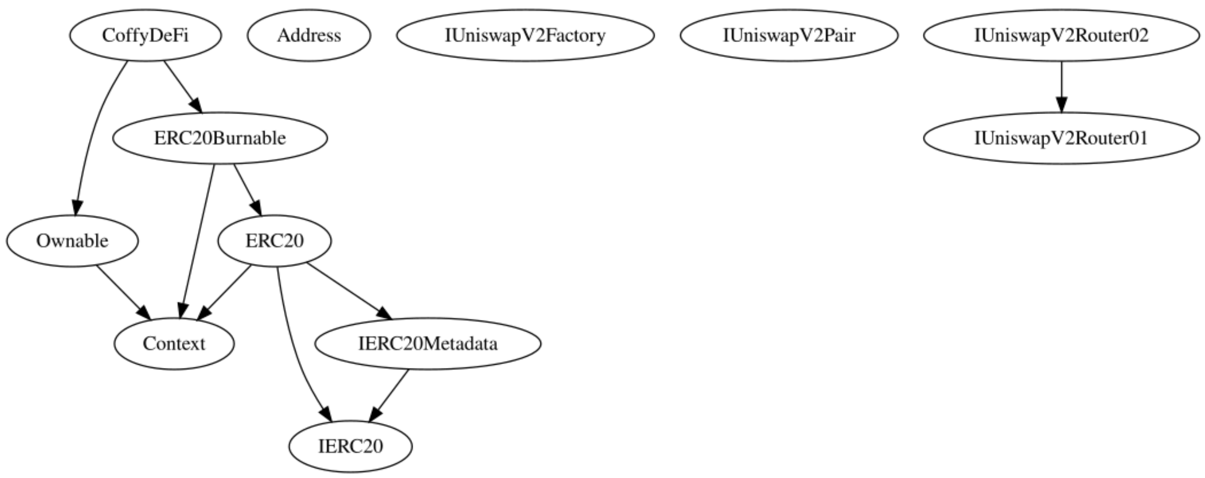
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	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	✓	-
CoffyDeFi	Implementation	ERC20Burnable, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	getBalance	Private		
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	setBurnFee	External	✓	onlyOwner
	setReflectionFee	Public	✓	onlyOwner
	setLiquidityPoolFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	setDeveloperFee	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setDeveloperAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner

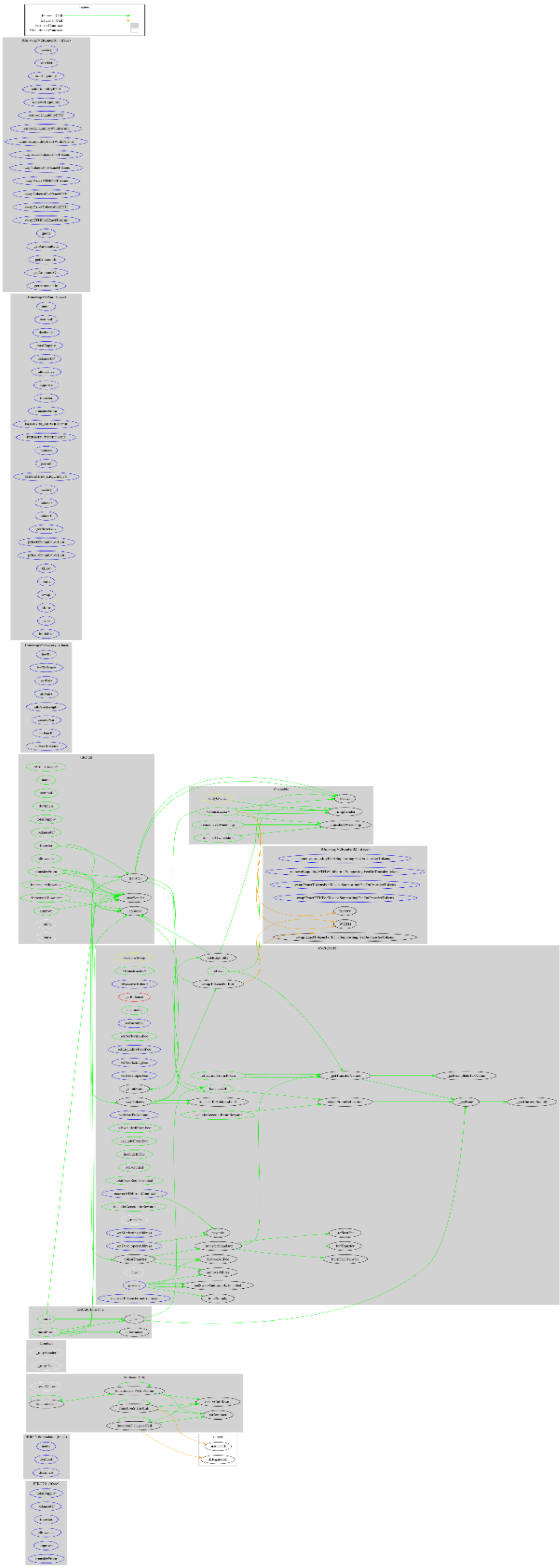
	setMaxTxAmount	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	isExcluded	Public		-
	totalFeesRedistributed	Public		-
	_mintStart	Private	✓	
	reflect	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Private		
	excludeAccountFromReward	Public	✓	onlyOwner
	includeAccountinReward	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_getTransferValues	Private		
	_getCompleteTaxValue	Private		
	_reflectFee	Private	✓	
	burnFeeTransfer	Private	✓	
	feeTransfer	Private	✓	
	_getRate	Private		
	_getCurrentSupply	Private		
	swapTokens	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	

	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	presale	External	✓	onlyOwner
	transferToAddressETH	Private	✓	
	_burn	Internal	✓	
	recoverETHfromContract	External	✓	onlyOwner
	recoverTokensFromContract	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

CoffyDefi contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no critical issues. The contract Owner can access some admin functions that cannot be used in a malicious way. There is also a limit of max 5% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io