Cyberscope

# Audit Report

# Coffy DeFi

September 2024

| | |
|---|---|
| Network | ETH |
| Address | 0x828FCF32729b35857e34F0a5315a8B892B220a34 |
| 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 |
|---|---|---|---|
| Critical | ZD | Zero Division | Unresolved |
| Medium | FRV | Fee Restoration Vulnerability | Unresolved |
| Medium | SEI | Self Exclusion Inconsistency | Unresolved |
| Minor / Informative | CCR | Contract Centralization Risk | Unresolved |
| Minor / Informative | DDP | Decimal Division Precision | Unresolved |
| Minor / Informative | IDI | Immutable Declaration Improvement | Unresolved |
| Minor / Informative | MEE | Missing Events Emission | Unresolved |
| Minor / Informative | PLPI | Potential Liquidity Provision Inadequacy | Unresolved |
| Minor / Informative | PTRP | Potential Transfer Revert Propagation | Unresolved |
| Minor / Informative | PVC | Price Volatility Concern | Unresolved |
| Minor / Informative | RED | Redundant Event Declaration | Unresolved |
| Minor / Informative | RRA | Redundant Repeated Approvals | Unresolved |
| Minor / Informative | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| Minor / Informative | 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

# 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

| | |
|---|---|
| **Contract Name** | CoffyDeFi |
| **Compiler Version** | v0.8.25+commit.b61c2a91 |
| **Optimization** | 200 runs |
| **Explorer** | https://etherscan.io/address/0x828fcf32729b35857e34f0a5315a8b892b220a34 |
| **Address** | 0x828fcf32729b35857e34f0a5315a8b892b220a34 |
| **Network** | ETH |
| **Symbol** | COFFI |
| **Decimals** | 18 |
| **Total Supply** | 1,000,000,000 |

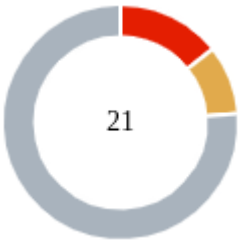
## Audit Updates

| | |
|---|---|
| **Initial Audit** | 19 Sep 2024 |

## Source Files

| | |
|---|---|
| **Filename** | SHA256 |
| **CoffyDeFi.sol** | c0a93bcd06a562b131faad9fdf8575458281ca65810775a20bcd1d1dc4920590 |

# Findings Breakdown




| | |
|---|---|
| Critical | 3 |
| Medium | 2 |
| Minor / Informative | 16 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 3 | 0 | 0 | 0 |
| Medium | 2 | 0 | 0 | 0 |
| Minor / Informative | 16 | 0 | 0 | 0 |

## ST - Stops Transactions

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | CoffyDeFi.sol#L1564 |
| **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** | CoffyDeFi.sol#L1390,1394,1403,1412 |
| **Status** | Unresolved |

## Description

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

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

function setLiquidityPoolFee(uint256 liquidityPoolFee_)
external onlyOwner {
        _liquidityPoolFee = liquidityPoolFee_;
        _combinedLiquidityFee = _liquidityPoolFee +
_marketingFee +
        _developerFee + _charityFee;
    }

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

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

function setCharityFee(uint256 charityFee_) external onlyOwner
{
        _charityFee = charityFee_;
        _combinedLiquidityFee = _liquidityPoolFee +
```

```
_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** | CoffyDeFi.sol#L1734,1745 |
| **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` variable and the term `(_combinedLiquidityFee * 10 - ((_liquidityPoolFee * 10) / 2))` can be set to zero.

```
uint256 lpTokenBalance = (contractTokenBalance *
_liquidityPoolFee) /
        _combinedLiquidityFee;
```

```
transferToAddressETH(
marketingAddress,
((transferredBalance) * (_marketingFee * 10)) /
(_combinedLiquidityFee * 10 - ((_liquidityPoolFee * 10) / 2)));

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

transferToAddressETH(
charityAddress,
((transferredBalance) * (_charityFee * 10)) /
(_combinedLiquidityFee * 10 - ((_liquidityPoolFee * 10) / 2)));

uint256 liquidityBalance = (transferredBalance *
((_liquidityPoolFee * 10) / 2)) /
((_combinedLiquidityFee * 10) - ((_liquidityPoolFee * 10) /
2));
```

## 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.

## FRV - Fee Restoration Vulnerability

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | CoffyDeFi.sol#L1597 |
| **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 (_combinedLiquidityFee == 0 && _reflectionFee == 0)
return;

    _previousCombinedLiquidityFee = _combinedLiquidityFee;
    _previousLiquidityPoolFee = _liquidityPoolFee;
    _previousBurnFee = _burnFee;
    _previousReflectionFee = _reflectionFee;
    _previousMarketingFee = _marketingFee;
    _previousDeveloperFee = _developerFee;
    _previousCharityFee = _charityFee;

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

function restoreAllFee() private {
    _combinedLiquidityFee = _previousCombinedLiquidityFee;
    _liquidityPoolFee = _previousLiquidityPoolFee;
    _burnFee = _previousBurnFee;
    _reflectionFee = _previousReflectionFee;
    _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.

## SEI - Self Exclusion Inconsistency

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | CoffyDeFi.sol#L1692 |
| **Status** | Unresolved |

## Description

The contract implements a fee mechanism that accrues fees on token transfers. The `feeTransfer` function is responsible for transferring the withheld fees from a transaction to the contract address. However, this function assumes that the contract address is not excluded from the reflection mechanism. If the contract address `_isExcluded` from the reflection mechanism, the function only updates the `_rOwned` balance. This has the potential to disrupt the consistency between the token's total supply and the sum of the individual balances.

```
function feeTransfer(
        address sender,
        uint256 tAmount,
        uint256 currentRate,
        uint256 fee,
        address receiver
    ) private {
        uint256 tFee = (tAmount * fee) / 100;
        if (tFee > 0) {
            uint256 rFee = tFee * currentRate;
            _rOwned[receiver] = _rOwned[receiver] + rFee;
            emit Transfer(sender, receiver, tFee);
        }
}
```

## Recommendation

The team is advised to ensure that the contract address cannot be excluded from the reflection mechanism or to revise the `feeTransfer` function to properly handle the case of self-exclusion. This adaptation will guarantee consistency between the total supply and individual balances.

# CCR - Contract Centralization Risk

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | CoffyDeFi.sol#L1390,1394,1403,1412,1421,1430,1434,1438,1442,1448,1453,1457,1461,1465,1477,1529,1538,1858 |
| **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 setLiquidityPoolFee(uint256 liquidityPoolFee_) external
onlyOwner {}

function setMarketingFee(uint256 marketingFee_) external onlyOwner {}

function setDeveloperFee(uint256 developerFee_) external onlyOwner {}

function setCharityFee(uint256 charityFee_) external onlyOwner {}

function setMarketingAddress(address _marketingAddress) external
onlyOwner {}

function setDeveloperAddress(address _developerAddress) external
onlyOwner {}

function setCharityAddress(address _charityAddress) external onlyOwner
{}

function setNumTokensSellToAddToLiquidity(
uint256 _minimumTokensBeforeSwap) 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 setReflectionFee(uint256 newReflectionFee) public onlyOwner {}

function excludeAccountFromReward(address account) public onlyOwner {}

function includeAccountinReward(address account) public onlyOwner {}

function presale(bool _presale) 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#L1745 |
| **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
)));

transferToAddressETH(
charityAddress,
((transferredBalance) * (_charityFee * 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#L1303 |
| **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#L1461,1465,1858 |
| **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** | CoffyDeFi.sol#L1774 |
| **Status** | Unresolved |

## Description

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

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

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

        // make the swap

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

        emit SwapTokensForETH(tokenAmount, path);
}
```

## Recommendation

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

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

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

## PTRP - Potential Transfer Revert Propagation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | CoffyDeFi.sol#L1871 |
| **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 {
    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** | CoffyDeFi.sol#L |
| **Status** | Unresolved |

### Description

The contract accumulates tokens from taxes to swap them for ETH. The variable `minimumTokensBeforeSwap` sets the threshold at which the contract triggers this swap. If this threshold is set to a high value, a large quantity of tokens will be swapped for ETH in a single transaction. Given the potential volatility of the token's price, executing a large swap at once could result in substantial price fluctuations. This may impact the value of the swap, leading to significant volatility in the token price and potentially affecting both the contract and market participants 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 - Redundant Event Declaration

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | CoffyDeFi.sol#L1277 |
| **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.

## RRA - Redundant Repeated Approvals

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | CoffyDeFi.sol#L1519 |
| **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 token transfer to cover all possible
scenarios
        _approve(address(this), address(uniswapV2Router),
tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            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.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | CoffyDeFi.sol#L954,956,986,1028,1246,1252,1255,1258,1261,1264,1267,1430,1434,1438,1443,1448,1905 |
| **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 _reflectionFee
uint256 public _liquidityPoolFee
uint256 public _burnFee
uint256 public _marketingFee
uint256 public _developerFee
uint256 public _charityFee
uint256 public _maxTxAmount
address _marketingAddress
address _developerAddress
address _charityAddress

...
```

## 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** | CoffyDeFi.sol#L707,741,764,1837 |
| **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");
...
        }
        _balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);

        _afterTokenTransfer(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** | CoffyDeFi.sol#L1337,1710,1711,1722,1724,1738,1740,1802 |
| **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 liquidityBalance = (transferredBalance *
            ((_liquidityPoolFee * 10) / 2)) /
            ((_combinedLiquidityFee * 10) - ((_liquidityPoolFee
* 10) / 2))
```

### Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | 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** | CoffyDeFi.sol#L362 |
| **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** | CoffyDeFi.sol#L7 |
| **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.

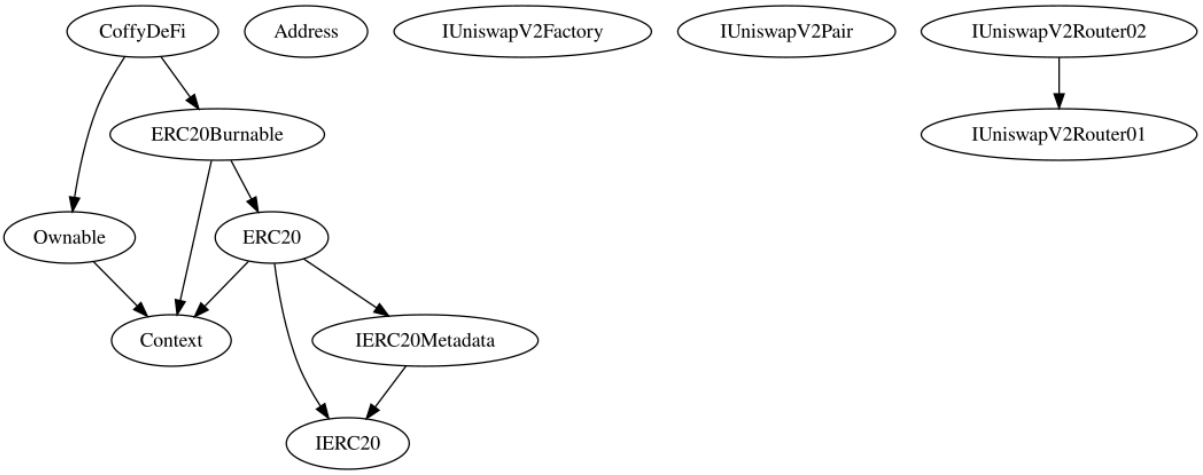
# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| **CoffyDeFi** | Implementation | ERC20Burnable, Ownable | | |
| | | Public | Payable | ERC20 |
| | | External | Payable | - |
| | getBalance | Private | | |
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | minimumTokensBeforeSwapAmount | Public | | - |
| | setBurnFee | External | ✓ | onlyOwner |
| | setLiquidityPoolFee | External | ✓ | onlyOwner |
| | setMarketingFee | External | ✓ | onlyOwner |
| | setDeveloperFee | External | ✓ | onlyOwner |
| | setCharityFee | External | ✓ | onlyOwner |
| | setMarketingAddress | External | ✓ | onlyOwner |
| | setDeveloperAddress | External | ✓ | onlyOwner |
| | setCharityAddress | External | ✓ | onlyOwner |
| | setNumTokensSellToAddToLiquidity | External | ✓ | onlyOwner |
| | setSwapAndLiquifyEnabled | Public | ✓ | onlyOwner |
| | setMaxTxAmount | External | ✓ | onlyOwner |
| | isExcludedFromFee | Public | | - |

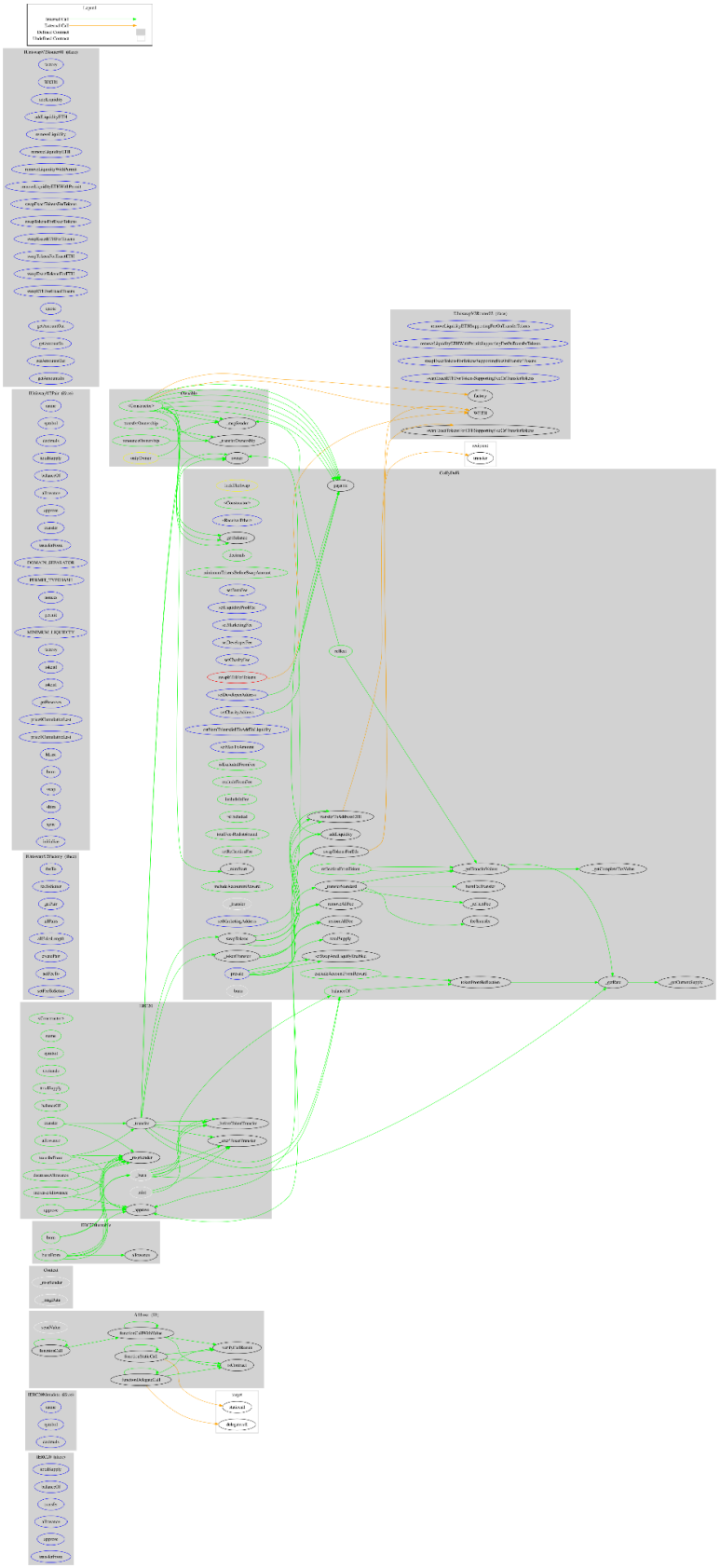
| | | | | |
|---|---|---|---|---|
| | excludeFromFee | Public | ✓ | onlyOwner |
| | includeInFee | Public | ✓ | onlyOwner |
| | isExcluded | Public | | - |
| | totalFeesRedistributed | Public | | - |
| | setReflectionFee | Public | ✓ | onlyOwner |
| | _mintStart | Private | ✓ | |
| | reflect | Public | ✓ | - |
| | 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 | ✓ | |
| | swapETHForTokens | Private | ✓ | |
| | addLiquidity | Private | ✓ | |

| | | | | |
|---|---|---|---|---|
| | removeAllFee | Private | ✓ | |
| | restoreAllFee | Private | ✓ | |
| | presale | External | ✓ | onlyOwner |
| | transferToAddressETH | Private | ✓ | |
| | _burn | Internal | ✓ | |

# Inheritance Graph



# Flow Graph



# Summary

CoffyDefi 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**

cyberscope.io