Cyberscope

Audit Report

COIF.CAPITAL

July 2023

Repository https://github.com/coifcapitalaudit/contract/tree/main

Commit 241376f803a0bafc643d876f9349d94c76b1c78f

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	US	Untrusted Source	Unresolved
Minor / Informative	PUV	Potential Underflow Vulnerability	Unresolved
Minor / Informative	UBT	Unchecked Balance Transfer	Unresolved
Minor / Informative	RCS	Redundant Code Statement	Unresolved
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	TUU	Time Units Usage	Unresolved
Minor / Informative	DKO	Delete Keyword Optimization	Unresolved
Minor / Informative	RSD	Redundant Swap Duplication	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	OCTD	Transfers Contract's Tokens	Unresolved
Minor / Informative	MC	Missing Check	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	MMN	Misleading Method Naming	Unresolved

●	MU	Modifiers Usage	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

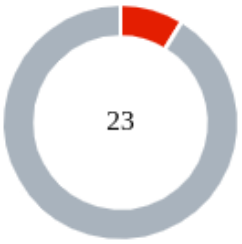
Initial Audit	07 Aug 2023

Source Files

Filename	SHA256
contracts/coif_contract_mainchain_audit_v04.sol	5a4b73f8bac0a356f38dd2644416fbd245fadf6416744319c72212fb89f752fc
contracts/lock_vesting_V01.sol	2566bdefb11655517a5c666c3852ccc5a39b9716c34d0d2fb48c539d6a012875
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol	29c75e69ce173ff8b498584700fef76bc81498c1d98120e2877a1439f0c31b5a
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	fc16aa4564878e1bb65740239d0c1422451cd32136306626ac37f5d5e0606a7b
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/ERC20.sol	d20d52b4be98738b8aa52b5bb0f88943f62128969b33d654fbca731539a7fe0a

@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	21

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	21	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/coif_contract_mainchain_audit_v04.sol#L516
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
        if(!tradingIsEnabled) {
            require(canTransferBeforeTradingIsEnabled[from],
"Error: This account cannot send tokens until trading is
enabled");
        }
```

Recommendation

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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

US - Untrusted Source

Criticality	Critical
Location	contracts/coif_contract_mainchain_audit_v04.sol#L176
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
   function updateDividendDistributor(address _newAddress)
public onlyOwner {
        require(_newAddress != address(0), "Error: Address
cannot be zero");
        require(_newAddress != address(poolDistributor),
"Error: Dividend distributor already has that address");
        DividendDistributor newPoolDistributor =
DividendDistributor(payable(_newAddress));
        require(newPoolDistributor.owner() == address(this),
"Error: The new dividend distributor must be owned by the token
contract");

        emit UpdateDividendDistributor(_newAddress,
poolDistributorAddress);
        poolDistributor = newPoolDistributor;
        poolDistributorAddress = address(poolDistributor);
        excludedFromDividends[poolDistributorAddress] = true;
    }
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

PUV - Potential Underflow Vulnerability

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L904
Status	Unresolved

Description

The contract handles the distribution of tokens to the `_poolDistributorAddress` through the payoutPool1TokensAmount function. However, a potential underflow vulnerability has been identified in the function. Specifically, if the balance of _poolDistributorAddress is larger than the _payoutPool1CurrentTokenAmount, the subtraction operation will underflow. This could occur if a malicious user sends balance to the _poolDistributorAddress, for instance. Underflows in Solidity can lead to unexpected results and potential exploits, as they cause the calculation to wrap around and start from the next largest possible value.

The contract contains a potential underflow issue in the token distribution process, in the `processPool1` function. Specifically, the balance of `processPool1TokenERC20` in the `_poolDistributorAddress` may exceed the intended amount if, for instance, a malicious user sends balance to the `_poolDistributorAddress`. If the balance of `_poolDistributorAddress` is larger than the _payoutPool1CurrentTokenAmount, the subtraction operation in the following line of code will underflow.

```
payoutPool1TokensAmount[_processPool1Token] =
payoutPool1TokensAmount[_processPool1Token].add(_payoutPool1CurrentTokenA
mount.sub(processPool1TokenERC20.balanceOf(_poolDistributorAddress)));
```

Recommendation

It is recommended to implement safeguards against underflow in the contract. It is recommended to add a check to ensure that `processPool1TokenERC20.balanceOf(_poolDistributorAddress)` is not

greater than `_payoutPool1CurrentTokenAmount` before performing the subtraction. This would prevent the underflow from occurring.

UBT - Unchecked Balance Transfer

Criticality	Minor / Informative
Location	contracts/lock_vesting_V01.sol#L77
Status	Unresolved

Description

The contract is calculating an `unreleased` amount of a specific `_token` and transfer it to a `beneficiary_` address through the `release` function. However, the current implementation does not check or verify if the contract has enough balance of `unreleased` tokens to transfer to the `beneficiary_` address. This could potentially lead to a situation where the contract attempts to transfer more tokens than it holds, which would result in a failed transaction and could disrupt the contract's intended functionality.

```
function release(IERC20 _token) public onlyOwner {
    uint256 unreleased = releasableAmount(_token);
    require(unreleased > 0, "Error: no tokens to release");
    released[address(_token)] =
released[address(_token)].add(unreleased);
    _token.safeTransfer(beneficiary_, unreleased);
    emit TokensReleased(_token, unreleased);
  }
```

Recommendation

It is recommended to add a check to ensure that the contract has enough balance of the `_token` before attempting to transfer it. This could be achieved by querying the `_token` balance of the contract and comparing it to the `unreleased` amount. If the contract's balance is less than the `unreleased` amount, the contract could revert the transaction or handle the discrepancy in a manner that aligns with the contract's intended functionality.

RCS - Redundant Code Statement

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L853
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract is using an `else` case that does not contain any executable code. This else case is redundant because it explicitly states that no action should be taken if the condition `shares[_shareholder].amount > 0` is not met. In Solidity, if a condition in an if statement is not met and there is no else clause, the program will simply continue execution without taking any action. Therefore, an else clause that does nothing is unnecessary and adds to the complexity and size of the contract without providing any functional benefit.

```
if(shares[_shareholder].amount > 0) {
                removeShareholder(_shareholder);
                totalShares =
totalShares.sub(shares[_shareholder].amount);
                shares[_shareholder].amount = 0;
            } else {
                /* no changes */
            }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

It is recommended to remove the redundant `else` clause from the contract. This will make the contract more concise and easier to read, without affecting the contract's functionality.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L574,1134
Status	Unresolved

Description

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

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

```
collectedAmountPool1Fee =
collectedAmountPool1Fee.add(pool1Fee);

payoutPool2TimeNext =  block.timestamp +
payoutPool2FrequencySec;
```

Recommendation

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

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L825
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specificallty, the contract is using repetitive code segments in the segment where `_processPool1Active` or `_processPool2Active` is true regarding the pool1 and pool2 operations. The code segments for `_processPool1Active` and `_processPool2Active` are almost identical, differing only in the specific pool they are processing.

```
            if (_processPool1Active){
                if (shareholderIndexes[_shareholder] <
_payoutPool1ShareholderCount) {
                    if (currentIndexPool1 <
shareholderIndexes[_shareholder]) {
                        if(shares[_shareholder].amount < _amountNew){
                            shares[_shareholder].amountExcludedBuyPool1 =
(_amountNew.sub(shares[_shareholder].amount)).add(shares[_shareholder].am
ountExcludedBuyPool1);
                        }
                    }
                }
            }

            if (_processPool2Active){
                if (shareholderIndexes[_shareholder] <
_payoutPool2ShareholderCount) {
                    if (currentIndexPool2 <
shareholderIndexes[_shareholder]) {
                        if(shares[_shareholder].amount < _amountNew) {
                            shares[_shareholder].amountExcludedBuyPool2 =
(_amountNew.sub(shares[_shareholder].amount)).add(shares[_shareholder].am
ountExcludedBuyPool2);
                        }
                    }
                }
            }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L808,1127
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
payoutPool2FrequencySec = 60*60*24*7*2;

    function updatePayoutPool2FrequencySec(uint256
_newPayoutPool2FrequencySec) external onlyOwner {
        emit
PayoutPool2FrequencySecUpdated(_newPayoutPool2FrequencySec,
payoutPool2FrequencySec);
        payoutPool2FrequencySec = _newPayoutPool2FrequencySec;
        updatePayoutPool2TimeNext();
    }
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L670
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
collectedAmountLiquidityFee = 0;
collectedAmountMarketingFee = 0;
collectedAmountPool1Fee = 0;
collectedAmountPool2Fee = 0;
collectedAmountPool3Fee = 0;
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L645
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
    if(_collectedAmountMarketingFeeDist > 0) {

        swapAndSendFeeWBNB(_collectedAmountMarketingFeeDist,
marketingWallet);
    }

    if(_collectedAmountPool1FeeDist > 0) {
        swapAndSendFeeWBNB(_collectedAmountPool1FeeDist,
pool1Wallet);
    }

    if(_collectedAmountPool2FeeDist > 0) {
        swapAndSendFeeWBNB(_collectedAmountPool2FeeDist,
poolDistributorAddress);
    }
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L253,522
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapFeeTokensMinAmount` 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.

```
   function setSwapFeeTokensMinAmount(uint256 _swapMinAmount)
public onlyOwner {
        require(_swapMinAmount <= (10**18), "Error: use the
value without 10**18, e.g. 10000 for 10000 tokens");
        swapFeeTokensMinAmount = _swapMinAmount.mul(10**18);
        emit SetSwapFeeTokensMinAmount(swapFeeTokensMinAmount);
    }
...
  if(
            tradingIsEnabled &&
            (balanceOf(address(this))>=swapFeeTokensMinAmount)
&&
            !feesSwapping &&
            !automatedMarketMakerPairs[from] &&
            !excludedFromFees[from] &&
            !excludedFromFees[to]
        ) {
            feesSwapping = true;
            distributeCollectedFees(
                collectedAmountLiquidityFee,
                collectedAmountMarketingFee,
                collectedAmountPool1Fee,
                collectedAmountPool2Fee,
                collectedAmountPool3Fee
            );
            feesSwapping = false;
        }
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L496,571
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract by transferring it to `pool1Wallet` wallet. The owner may take advantage of it by calling the `transferERC20TokenFromContractAddressToPool1` and `transferBNBFromContractAddressToPool1` functions.

Additionally, an inconsistency may occure between the contract balance and the accumulated fees. The variables `collectedAmountLiquidityFee`, `collectedAmountMarketingFee`, `collectedAmountPool1Fee`, `collectedAmountPool2Fee`, and `collectedAmountPool3Fee` are designed to accumulate tokens from fees. However, the token contract can be withdrawn from the contract and in this case, the accumulated fee variables are not initialized.

```
    function transferERC20TokenFromContractAddressToPool1(address
_tokenERC20) public onlyOwner {
            ERC20 tokenERC20 = ERC20(_tokenERC20);
            uint256 amount = tokenERC20.balanceOf(address(this));
            tokenERC20.transfer(pool1Wallet, amount);
    }

    function transferBNBFromContractAddressToPool1() public onlyOwner {
            address payable pool1WalletBNB = payable(pool1Wallet);
            pool1WalletBNB.transfer(address(this).balance);
    }

        collectedAmountLiquidityFee =
collectedAmountLiquidityFee.add(liquidityFee);
        collectedAmountMarketingFee =
collectedAmountMarketingFee.add(marketingFee);
        collectedAmountPool1Fee = collectedAmountPool1Fee.add(pool1Fee);
        collectedAmountPool2Fee = collectedAmountPool2Fee.add(pool2Fee);
        collectedAmountPool3Fee = collectedAmountPool3Fee.add(pool3Fee);
```

Recommendation

It is recommended to implement a mechanism that updates the balance when tokens are withdrawn from the contract. This can be achieved by adjusting the accumulated fee variables whenever a withdrawal is made. Alternatively, the contract could be modified to disallow the withdrawal of the token from the contract's address. This would ensure that the balance of the contract and the accumulated fees remain consistent, reducing the potential for exploitation and improving the overall security and reliability of the contract.

Additionally, 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

MC - Missing Check

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L253,385,1135
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, the contract contains the function, `setMinimumBalanceForDividends` `setSwapFeeTokensMinAmount` and `updateMinimumTokenBalanceForDividends`, which accept a token amount as a parameter. However, the contract does not have any checks in place to prevent these parameters from exceeding the total supply of tokens, which is set at `_totalSupply = 100_000_000`. This lack of checks allows for the possibility of passing a token amount larger than the total supply to these functions, which could lead to unexpected behavior or potential vulnerabilities in the contract.

```
    function setSwapFeeTokensMinAmount(uint256 _swapMinAmount) public
onlyOwner {
        require(_swapMinAmount <= (10**18), "Error: use the value
without 10**18, e.g. 10000 for 10000 tokens");
        swapFeeTokensMinAmount = _swapMinAmount.mul(10**18);
        emit SetSwapFeeTokensMinAmount(swapFeeTokensMinAmount);
    }

    function setMinimumBalanceForDividends(uint256 _newMinimumBalance)
public onlyOwner {
        require((!processPool1Active && !processPool2Active), "Error:
process pool1 payout or pool2 payout is active, wait till end");

poolDistributor.updateMinimumTokenBalanceForDividends(_newMinimumBalance
);
    }

    function updateMinimumTokenBalanceForDividends(uint256
_newMinimumBalance) external onlyOwner {
        require(_newMinimumBalance <= (10**18), "Error: use the value
without 10**18, e.g. 100 tokens");
        minimumTokenBalanceForDividends =
_newMinimumBalance.mul(10**18);
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications. It is recommended to implement checks within the `updateMinimumTokenBalanceForDividends` and `setSwapFeeTokensMinAmount` functions to ensure that the token amount passed as a parameter does not exceed the total supply of tokens. This can be achieved by adding a require statement in each function that compares the parameter to the total supply. If the parameter exceeds the total supply, the require statement should revert the transaction. This will prevent the possibility of setting these variables to a value larger than the total supply, thereby enhancing the security and predictability of the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L266,275
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the state of excludedFromFees addresses even if their current state is the same as the the one passed as an argument. As a result, the contract performs redundant storage writes.

```
    function excludeFromFees(address _account, bool _excluded)
public onlyOwner {
        excludedFromFees[_account] = _excluded;
        emit ExcludeFromFees(_account, _excluded);
    }

    function excludeMultipleAccountsFromFees(address[] calldata
_accounts, bool _excluded) public onlyOwner {
        for(uint256 i = 0; i < _accounts.length; i++) {
            excludedFromFees[_accounts[i]] = _excluded;
        }
        emit ExcludeMultipleAccountsFromFees(_accounts,
_excluded);
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L218,1152
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

Specifically,the contract is utilizing the `pool3BurnAddress` to distribute tokens. This address is initially set to the zero address, but the contract owner has the ability to change this address by calling the `setPool3BurnAddress` function. This could potentially lead to confusion or misuse, as the `pool3BurnAddress` may no longer represent a burn address but could be set to any arbitrary address. This is particularly concerning because the function names `setPool3BurnAddress` and `updatePool3BurnAddress` suggest that these addresses are specifically for burning tokens, but the implementation allows for a broader range of functionality.

Furthermore, the `updatePool3BurnAddress` function emits an event `Pool3BurnAddressUpdated` whenever the burn address is updated. This event could be misleading if the updated address is not actually a burn address. This discrepancy between the function and variable names and their actual functionality can lead to confusion and misinterpretation of the contract's behavior, making the code more difficult to read and understand.

```
  function setPool3BurnAddress(address _newPool3BurnAddress)
public onlyOwner {
        require(_newPool3BurnAddress != address(0), "Error:
Address cannot be zero");

poolDistributor.updatePool3BurnAddress(_newPool3BurnAddress);
    }

    function updatePool3BurnAddress(address
_newPool3BurnAddress) external onlyOwner {
        require(_newPool3BurnAddress != pool3BurnAddress,
"Error: The pool3BurnAddress is already this address");
        emit Pool3BurnAddressUpdated(_newPool3BurnAddress,
pool3BurnAddress);
        pool3BurnAddress = _newPool3BurnAddress;
    }
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. It is recommended to rename both the function and the variable to more accurately reflect their actual implementation.This would make it clear that these addresses are used for distribution, not necessarily for burning. Additionally, the event `Pool3BurnAddressUpdated` could be renamed to avoid any confusion. By doing so, the contract would become more self-explanatory and easier to understand, reducing the risk of misuse or misinterpretation.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L177,189,197,205,213,219,224,229,234,239,244,249
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_newAddress != address(0), "Error: Address cannot be zero");
require(_newLiquidityWallet != address(0), "Error: Address cannot be zero");
require(_newMarketingWallet != address(0), "Error: Address cannot be zero");
require(_newPool1Wallet != address(0), "Error: Address cannot be zero");
require(_newPool3Wallet != address(0), "Error: Address cannot be zero");
require(_newPool3BurnAddress != address(0), "Error: Address cannot be zero");
require(_newTeamWallet != address(0), "Error: Address cannot be zero");
require(_newLongTermGrowthWallet != address(0), "Error: Address cannot be zero");
require(_newEcosystemWallet != address(0), "Error: Address cannot be zero");
require(_newTeamLockAddress != address(0), "Error: Address cannot be zero");
require(_newLongTermGrowthLockAddress != address(0), "Error: Address cannot be zero");
require(_newEcosystemLockAddress != address(0), "Error: Address cannot be zero");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.solcontracts/lock_vesting_V01.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

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

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L807
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.

```
dividendsPerShareAccuracyFactor
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L50,762
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.

```
ERC20 internal WBNB =
ERC20(0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c)
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L50,54,120,176,188,196,204,212,218,223,228,233,238,243,248,253,259,266,271,275,282,298,310,315,316,317,318,319,329,330,331,332,333,343,344,345,346,347,356,361,368,377,385,390,395,453,466,486,492,496,629,630,631,632,633,678,691,706,722,762,811,812,813,814,815,816,857,862,872,873,874,882,883,884,885,886,887,929,930,931,932,963,964,965,966,1005,1006,1007,1046,1058,1081,1103,1119,1125,1135,1144,1150,1156,1162,1168,1174,1180,1186contracts/lock_vesting_V01.sol#L69,73,81,107
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.

```
ERC20 internal WBNB =
ERC20(0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c)
uint8 private constant _decimals = 18
event isExcludeFromDividends(address indexed account, bool
isExcluded);
address _newAddress
address _newLiquidityWallet
address _newMarketingWallet
address _newPool1Wallet
address _newPool3Wallet
address _newPool3BurnAddress
address _newTeamWallet
address _newLongTermGrowthWallet
address _newEcosystemWallet
address _newTeamLockAddress
address _newLongTermGrowthLockAddress
...
```

```
address _token
IERC20 _token
address _newBeneficiary
```

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	contracts/coif_contract_mainchain_audit_v04.sol#L321,335,349,400,843
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.

```
buyLiquidityFee = _newBuyLiquidityFee
sellLiquidityFee = _newSellLiquidityFee
txLiquidityFee = _newTxLiquidityFee
processPool1StartTime = _startTime
totalShares =
totalShares.sub(shares[_shareholder].amount).add(_amountNew)
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L615,617,1121,1122
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 _divPerShare =
(WBNB.balanceOf(_poolDistributorAddress)).mul(dividendsPerShareAccuracyFa
ctor).div(totalShares)
return
(shares[_shareholder].amount).mul(_divPerShare).div(dividendsPerShareAccu
racyFactor)
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/coif_contract_mainchain_audit_v04.sol#L499,878,985,988,991,994,997,1026,1029,1032,1035,1038
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
tokenERC20.transfer(pool1Wallet, amount)
pool1TokenERC20.transfer(_pool1Wallet, amount)
processPool1TokenERC20.transfer(pool3Wallet, amount)
processPool1TokenERC20.transfer(teamWallet, amount)
processPool1TokenERC20.transfer(longTermGrowthWallet, amount)
processPool1TokenERC20.transfer(ecosystemWallet, amount)
processPool1TokenERC20.transfer(_shareholder, amount)
WBNB.transfer(pool3Wallet, amount)
WBNB.transfer(teamWallet, amount)
WBNB.transfer(longTermGrowthWallet, amount)
WBNB.transfer(ecosystemWallet, amount)
WBNB.transfer(_shareholder, amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IDividendDistributor	Interface			
	setShare	External	✓	-
	transferTokenFromPool2ToPool1	External	✓	-
	processPool1	External	✓	-
	processPool2	External	✓	-
CommunityInvestmentFundContract	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateDividendDistributor	Public	✓	onlyOwner
	setLiquidityWallet	Public	✓	onlyOwner
	setMarketingWallet	Public	✓	onlyOwner
	setPool1Wallet	Public	✓	onlyOwner
	setPool3Wallet	Public	✓	onlyOwner
	setPool3BurnAddress	Public	✓	onlyOwner
	setTeamWallet	Public	✓	onlyOwner
	setLongTermGrowthWallet	Public	✓	onlyOwner

	setEcosystemWallet	Public	✓	onlyOwner
	setTeamLockAddress	Public	✓	onlyOwner
	setLongTermGrowthLockAddress	Public	✓	onlyOwner
	setEcosystemLockAddress	Public	✓	onlyOwner
	setSwapFeeTokensMinAmount	Public	✓	onlyOwner
	updateUniswapV2Router	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	isExcludedFromFees	Public		-
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromDividends	Public	✓	onlyOwner
	isExcludedFromDividends	Public		-
	updateBuyFees	Public	✓	onlyOwner
	updateSellFees	Public	✓	onlyOwner
	updateTxFees	Public	✓	onlyOwner
	setTradeFeeStatus	Public	✓	onlyOwner
	setPayoutGas	Public	✓	onlyOwner
	setPayoutPool2Percent	Public	✓	onlyOwner
	setPayoutPool2MinAmountWBNB	Public	✓	onlyOwner
	setMinimumBalanceForDividends	Public	✓	onlyOwner
	setPayoutPool2FrequencySec	Public	✓	onlyOwner
	triggerPool1Payout	Public	✓	onlyOwner

	getCurrentInfoAboutPool1	Public		-
	getCurrentInfoAboutPool2	Public		-
	getAccountDividendsInfoForPool2	Public		-
	getAccountDividendsInfoForPool2AtIndex	Public		-
	launch	Public	✓	onlyOwner
	setCanTransferBeforeTradingIsEnabled	Public	✓	onlyOwner
	transferERC20TokenFromPool2ToPool1	Public	✓	onlyOwner
	transferERC20TokenFromContractAddressToPool1	Public	✓	onlyOwner
	transferBNBFromContractAddressToPool1	Public	✓	onlyOwner
	_transfer	Internal	✓	
	distributeCollectedFees	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForBNB	Private	✓	
	swapAndSendFeeWBNB	Private	✓	
	addLiquidity	Private	✓	
	getCollectedFeeAmounts	Public		-
DividendDistributor	Implementation	IDividendDistributor, Ownable		
		Public	✓	-
	setShare	External	✓	onlyOwner
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	

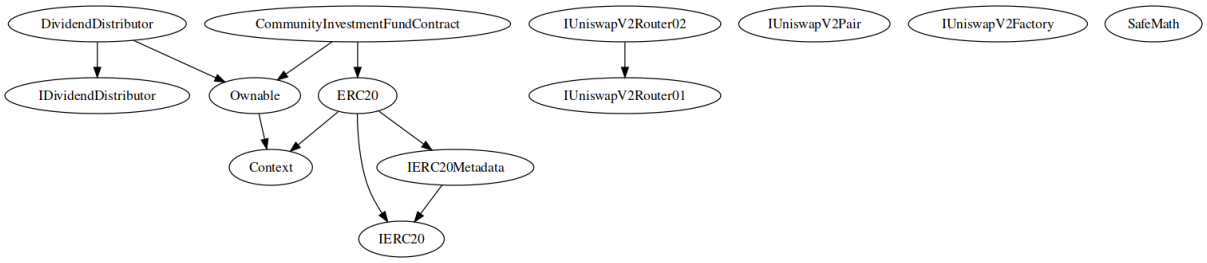
	transferTokenFromPool2ToPool1	External	✓	onlyOwner
	processPool1	External	✓	onlyOwner
	processPool2	External	✓	onlyOwner
	payoutDividendsPool1	Internal	✓	
	payoutDividendsPool2	Internal	✓	
	getInfoAboutPool1AtIndex	External		-
	getInfoAboutPool1AtToken	External		-
	getInfoAboutPool2	External		-
	getAccountInfoForPool2	Public		-
	getAccountInfoForPool2AtIndex	External		-
	getUnpaidDividendsFromPool2	Public		-
	updatePayoutPool2FrequencySec	External	✓	onlyOwner
	updatePayoutPool2TimeNext	Public	✓	onlyOwner
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	getNumberOfTokenHolders	External		-
	updatePool3Wallet	External	✓	onlyOwner
	updatePool3BurnAddress	External	✓	onlyOwner
	updateTeamWallet	External	✓	onlyOwner
	updateLongTermGrowthWallet	External	✓	onlyOwner
	updateEcosystemWallet	External	✓	onlyOwner
	updateTeamLockAddress	External	✓	onlyOwner
	updateLongTermGrowthLockAddress	External	✓	onlyOwner
	updateEcosystemLockAddress	External	✓	onlyOwner

TokenVesting	Implementation	Ownable		
		Public	✓	-
	beneficiary	Public		-
	start	Public		-
	lockDuration	Public		-
	vestingDuration	Public		-
	getLockEnd	Public		-
	getVestingEnd	Public		-
	startVestingNow	Public	✓	onlyOwner
	releasedTokens	Public		-
	release	Public	✓	onlyOwner
	releasableAmount	Public		-
	_vestedAmount	Private		
	updateBeneficiary	External	✓	onlyOwner
LongTermGrowthTokenVesting	Implementation	TokenVesting		
		Public	✓	TokenVesting
TeamTokenVesting	Implementation	TokenVesting		
		Public	✓	TokenVesting
EcosystemTokenVesting	Implementation	TokenVesting		

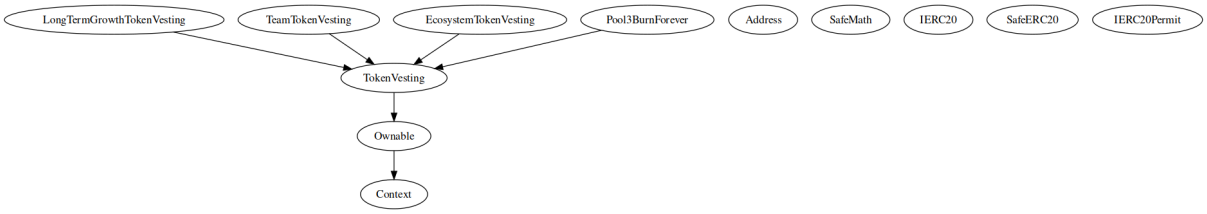
		Public	✓	TokenVesting
Pool3BurnForever	Implementation	TokenVesting		
		Public	✓	TokenVesting

Inheritance Graph

CommunityInvestmentFundContract

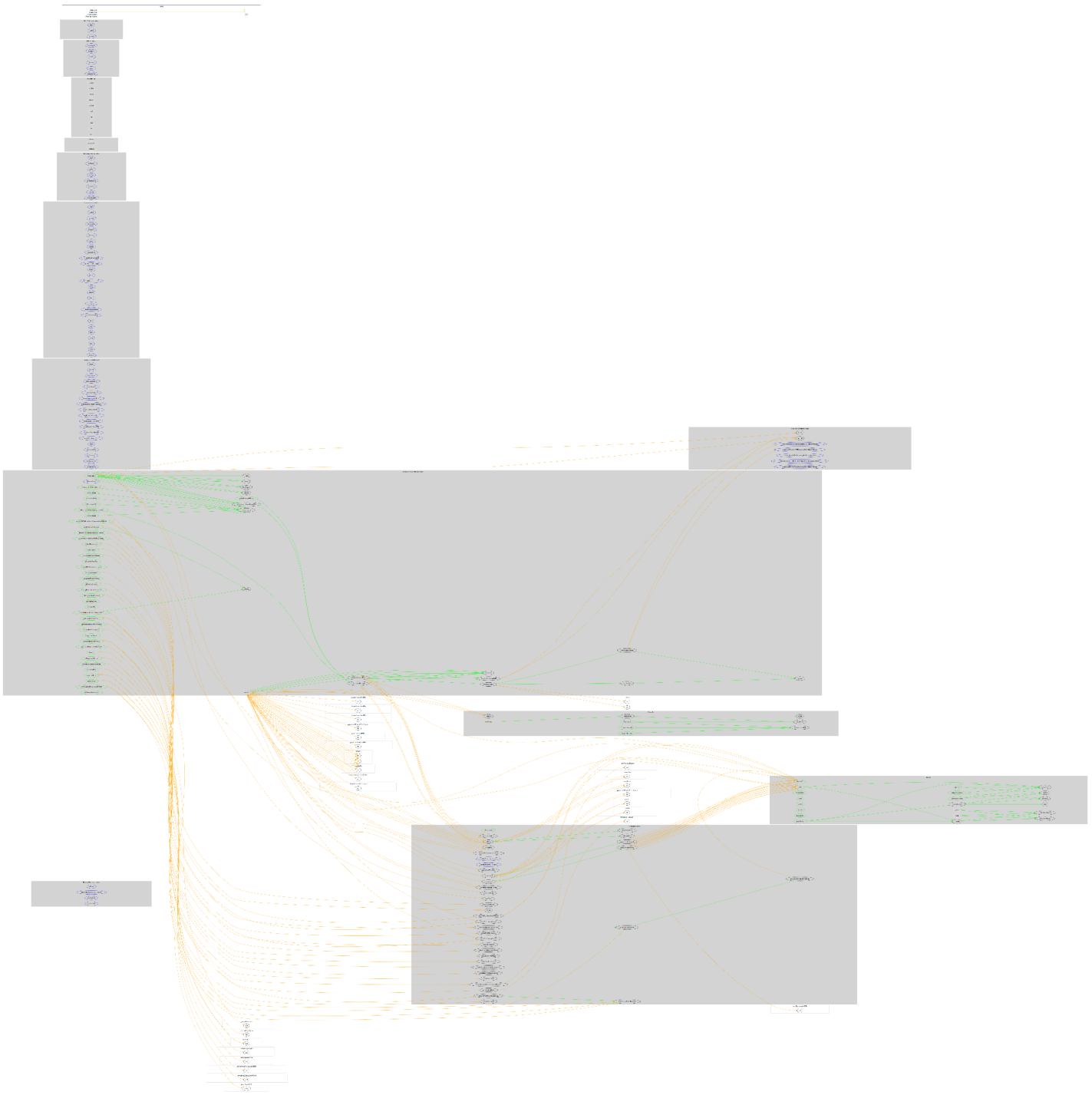


TokenVesting

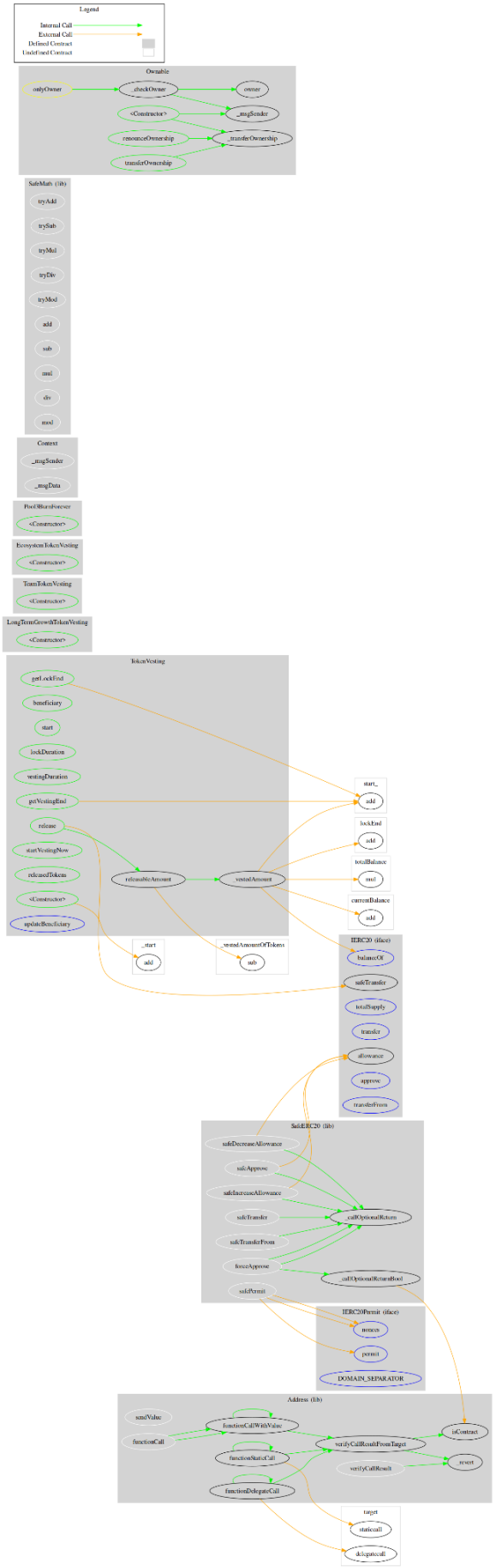


Flow Graph

CommunityInvestmentFundContract



TokenVesting



Summary

COIF.CAPITAL contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. COIF.CAPITAL is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 15% fees.

The TokenVesting contract implements a token vesting mechanism, locking all ERC20 tokens sent to it for a specified "lockDuration", followed by a linear vesting period defined by "vestingDuration".

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