Cyberscope

Audit Report

COIF.CAPITAL

Aug 2023

Repository	https://github.com/coifcapitalaudit/contract/tree/main
Commit	f610f3c9171174a2ba0153451f8871a466e1d4e5
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Acknowledged
●	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
●	MMN	Misleading Method Naming	Acknowledged
●	RSW	Redundant Storage Writes	Acknowledged
●	OCTD	Transfers Contract's Tokens	Acknowledged
●	PVC	Price Volatility Concern	Acknowledged
●	RSD	Redundant Swap Duplication	Acknowledged
●	CR	Code Repetition	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L07	Missing Events Arithmetic	Acknowledged
●	L13	Divide before Multiply Operation	Acknowledged
●	L20	Succeeded Transfer Check	Acknowledged
●	RSML	Redundant SafeMath Library	Acknowledged

Table of Contents

Review

Repository	https://github.com/coifcapitalaudit/contract/commits/main
Commit	f610f3c9171174a2ba0153451f8871a466e1d4e5
Testing Deploy - CommunityInvestmentFundContract.sol	https://testnet.bscscan.com/address/0xb1bcf69bcb0ef959b0aa23a6993abf84df25dd2c
Symbol	COIF
Decimals	18
Total Supply	100,000,000
Testing Deploy - TokenVesting.sol	https://testnet.bscscan.com/address/0xe9f1bef36d454c709a471de4a6522044b1f4bc88

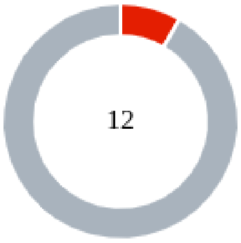
Audit Updates

Initial Audit	07 Aug 2023 https://github.com/cyberscope-io/audits/blob/main/coif/v1/audit.pdf
Corrected Phase 2	11 Aug 2023

Source Files

Filename	SHA256
contracts/coif_contract_mainchain_audit_v05.sol	5a4b73f8bac0a356f38dd2644416fbd245fadf6416744319c72212fb89f752fc
contracts/lock_vesting_V02.sol	af8db863694f5828a4432def9ecfa272ad75849785fa28c4b3d1066d6a17dfbf
@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

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	11

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	1	0	0
●	Medium	0	0	0	0
●	Minor / Informative	0	11	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/CommunityInvestmentFundContract.sol#L503
Status	Acknowledged

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 enabled 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

It is recommended to invoke the `launch` function, which sets the `tradingIsEnabled` variable to true. This action will effectively enable trading for all users. 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 multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Team Update

The team has acknowledged that this is not a security issue and states:

"*The procedure will be as follows: Deployer of contract can distribute the tokens at the beginning according to the allocation table. Only the Deployer and public and private sale contracts get the rights to transfer tokens. After the public sale, liquidity is added according to the allocation table and trading is activated immediately. The reason for "Owner needs to*

Enable Trading" is to avoid that no one adds liquidity at the unfavourable rate and thus makes it inconvenient to add inital liquidity at planned rate. Without liquidity, trading will not be possible. And as soon as liquidity is added, trading is also activated immediately. This will be communicated accordingly and can be checked on the blockchain."

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L207,1143
Status	Acknowledged

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 validateAddressNotZero(_newPool3BurnAddress) {
        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.

Team Update

The team has acknowledged that this is not a security issue and states:

"In this case, `pool3BurnAddress` is a vesting contract with lockDuration with max value. `pool3BurnAddress` is used as a "special" burn address as it is not possible to release the tokens once they have been sent to this vesting contract. After deployment of COIF contract pool3BurnAddress will be updated to this special vesting contract. This is done only once.

Please see an example for old deployed COIF contract: https://bscscan.com/address/0xb2ca5c351242c2e526fc5d89656848ce3511a79e#code contract Pool3BurnForever is TokenVesting { //lockDuration = near to max value of uint256, vestingDuration = 1 (zero not possible) uint256 private pool3LockDuration = type(uint256).max - block.timestamp 2; constructor () TokenVesting(address(0xfD92637A67cfCbd7eE0c79056325e3701123d5A2), block.timestamp, pool3LockDuration, 1) { } }*

For sure, there are other possibilities to implement special burn functionality.But there is no inconsistency between the function and variable names and their actual functionality in this case."

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L249,258
Status	Acknowledged

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.

Team Update

The team has acknowledged that this is not a security issue and states:

"*Since these functions are typically not called under normal circumstances (only in exceptional cases), the code does not need to be modified with respect to the efficiency and performance of the source code, as well as reducing the execution costs. Additional checks would only increase complexity without enhancing security.*"

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L483,559
Status	Acknowledged

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.

Additionaly, 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 + (liquidityFee);
        collectedAmountMarketingFee =
collectedAmountMarketingFee + (marketingFee);
        collectedAmountPool1Fee = collectedAmountPool1Fee +
(pool1Fee);
        collectedAmountPool2Fee = collectedAmountPool2Fee +
(pool2Fee);
        collectedAmountPool3Fee = collectedAmountPool3Fee +
(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.

Team Update

The team has acknowledged that this is not a security issue and states:

"*The function transferERC20TokenFromContractAddressToPool1 is designed to rescue the tokens that were sent to the contract address “CommunityInvestmentFundContract” accidentally. This does not apply to COIF tokens. The function is not used to withdraw COIF from the contract address. If someone accidentally sends COIF to the contract address and the amount exceeds 10,000 COIF, the function distributeCollectedFees will be triggered with the next sell/transfer, and all COIF tokens will be distributed. There will be no remaining COIF tokens that could be withdrawn. transferERC20TokenFromContractAddressToPool1 is an emergency function that is typically not executed.*"

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L235,508
Status	Acknowledged

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");
        require(_swapMinAmount <= 100_000_000, "Error: token
amount exceeds total supply");
        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.

Team Update

The team has acknowledged that this is not a security issue and states:

"The default value of `swapFeeTokensMinAmount` is set to 10,000 COIF. This corresponds to 0.01% of the total supply (100,000,000). The team will further reduce the value as the price increases. There's no need to make the code more complex."

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L632
Status	Acknowledged

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.

Team Update

The team has acknowledged that this is not a security issue and states:

"If we were to do the token swap operation once, we would need to convert WBNB to the individual fee parts. The code readability will be significantly worsened. In this particular case, the gas savings will not be significant. And since this part of the code isn't executed on every transfer, but only when a certain threshold of accumulated fees is reached, the code can remain in the more readable form. In addition, it was planned that the MarketingWallet would get a better conversion rate compared to Pool 1, Pool 2 would get the worst rate. This is also guaranteed with the current code because the rate deteriorates with each swap."

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L808
Status	Acknowledged

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.

Specifically, 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 - shares[_shareholder].amount) +
shares[_shareholder].amountExcludedBuyPool1;
                        }
                    }
                }
            }

 if (_processPool2Active){
                if (shareholderIndexes[_shareholder] <
_payoutPool2ShareholderCount) {
                    if (currentIndexPool2 <
shareholderIndexes[_shareholder]) {
                        if(shares[_shareholder].amount < _amountNew) {
                            shares[_shareholder].amountExcludedBuyPool2
= (_amountNew - shares[_shareholder].amount) +
shares[_shareholder].amountExcludedBuyPool2;
                        }
                    }
                }
            }
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

"It will become significantly more complex if we were to extract this portion of the code into an internal function, as it would require passing several parameters (for Pool 1 or Pool 2) as

well: _processPool1Active, _payoutPool1ShareholderCount, currentIndexPool1, amountExcludedBuyPool1"

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L52,118,127,128,129,133,181,188,195,202,207,211,215,219,223,227,231,235,242,249,254,258,265,281,293,298,299,300,301,302,313,314,315,316,317,328,329,330,331,332,342,347,354,363,371,376,381,440,453,473,479,483,616,617,618,619,620,665,678,693,709,796,797,798,799,800,801,840,845,855,856,857,865,866,867,868,869,870,921,922,923,924,955,956,957,958,997,998,999,1038,1050,1073,1095,1111,1117,1127,1137,1143,1149,1155,1161,1167,1173,1179contracts/TokenVesting.sol#L69,73,90,116
Status	Acknowledged

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.

```
uint8 private constant _decimals = 18
event isExcludeFromDividends(address indexed account, bool
isExcluded);
event updatedBuyFees(uint256 newBuyLiquidityFee, uint256
newBuyMarketingFee, uint256 newBuyPool1Fee, uint256
newBuyPool2Fee, uint256 newBuyPool3Fee);
event updatedSellFees(uint256 newSellLiquidityFee, uint256
newSellMarketingFee, uint256 newSellPool1Fee, uint256
newSellPool2Fee, uint256 newSellPool3Fee);
event updatedTxFees(uint256 newTxLiquidityFee, uint256
newTxMarketingFee, uint256 newTxPool1Fee, uint256
newTxPool2Fee, uint256 newTxPool3Fee);
event triggeredPool1Payout(bool indexed newProcessPool1Trigger,
address indexed newProcessPool1Token, uint256 indexed
newProcessPool1StartTime);
address _newLiquidityWallet
address _newMarketingWallet
address _newPool1Wallet
address _newPool3Wallet
address _newPool3BurnAddress
address _newTeamWallet
address _newLongTermGrowthWallet
address _newEcosystemWallet

...
```

```
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.

Team Update

The team has acknowledged that this is not a security issue and states:

"*Modifying the current contract would require too many changes. The risk of introducing errors in the process is too high.*"

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L828
Status	Acknowledged

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.

```
totalShares = totalShares - shares[_shareholder].amount +
_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.

Team Update

The team has acknowledged that this is not a security issue and states:

"*No event necessary because this function is updated after every transfer (buy/sell/tx): function* `setShare` "

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L602,604,1113,1114
Status	Acknowledged

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.

```
payoutPool2CurrentWBNB = (pool2BalanceWBNB *
payoutPool2Percent) / 100
payoutPool2DividendsPerShare = (payoutPool2CurrentWBNB *
poolDistributor.dividendsPerShareAccuracyFactor()) /
poolDistributor.totalShares()
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

"*First multiply then divide.*"

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/CommunityInvestmentFundContract.sol#L486,861,977,980,983,986,989,1018,1021,1024,1027,1030
Status	Acknowledged

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.

Team Update

The team has acknowledged that this is not a security issue and states:

"It's possible that a _shareholder is a smart contract (not an EOA) that doesn't allow transfers of WBNB or specific ERC20 tokens. In this case the method “safeTransfer” would stop processPool1 and processPool2 by revert of transaction. To ensure the process continues, we would need to manually exclude the problematic account from receiving

dividends. However, we would like to avoid doing that. To ensure that processPool1 and processPool2 don't fail (no revert!), we have to keep using „transfer“ method.

In addition, safeTransfer from SafeERC20 typically consumes more gas than the regular transfer from ERC20. This is because safeTransfer performs additional checks to ensure the transfer is successful before executing. These additional checks increase the gas consumption cost compared to directly using transfer. And we should try to use less gas during processPool1 and processPool2 execution."

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/TokenVesting.sol
Status	Acknowledged

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.

Team Update

The team has acknowledged that this is not a security issue and states:

"SafeMath library has been removed from coif_contract_mainchain_audit_v05.sol Since the vesting contract (lock_vesting_V02.sol) does not affect the COIF functionality, SafeMath can continue to be used here without any restrictions."

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	-
	setLiquidityWallet	Public	✓	onlyOwner validateAddressNotZero
	setMarketingWallet	Public	✓	onlyOwner validateAddressNotZero
	setPool1Wallet	Public	✓	onlyOwner validateAddressNotZero
	setPool3Wallet	Public	✓	onlyOwner validateAddressNotZero
	setPool3BurnAddress	Public	✓	onlyOwner validateAddressNotZero

	setTeamWallet	Public	✓	onlyOwner validateAddressNotZero
	setLongTermGrowthWallet	Public	✓	onlyOwner validateAddressNotZero
	setEcosystemWallet	Public	✓	onlyOwner validateAddressNotZero
	setTeamLockAddress	Public	✓	onlyOwner validateAddressNotZero
	setLongTermGrowthLockAddress	Public	✓	onlyOwner validateAddressNotZero
	setEcosystemLockAddress	Public	✓	onlyOwner validateAddressNotZero
	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		-

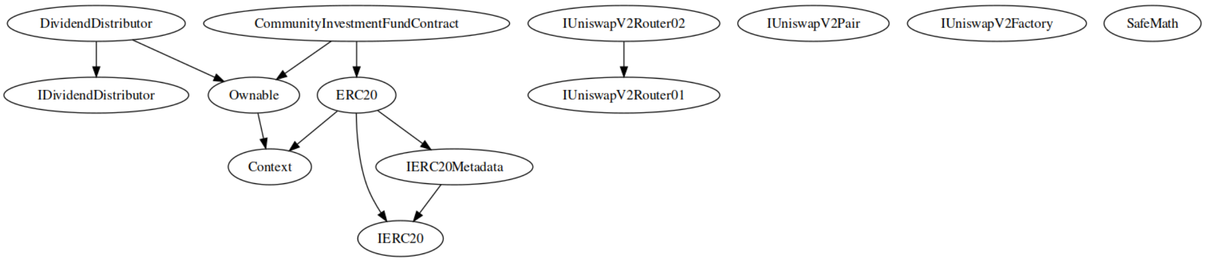
DividendDistributor	Implementation	IDividendDistributor, Ownable		
		Public	✓	-
	setShare	External	✓	onlyOwner
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
	transferTokenFromPool2ToPool1	External	✓	onlyOwner
	processPool1	External	✓	onlyOwner
	processPool2	External	✓	onlyOwner
	payoutDividendsPool1	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		

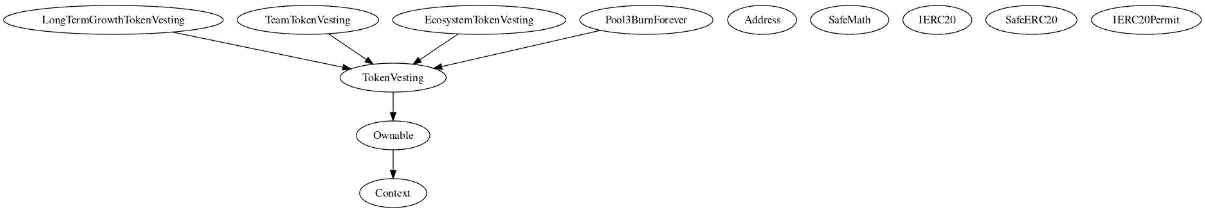
		Public	✓	TokenVesting
TeamTokenVesting	Implementation	TokenVesting		
		Public	✓	TokenVesting
EcosystemTokenVesting	Implementation	TokenVesting		
		Public	✓	TokenVesting
Pool3BurnForever	Implementation	TokenVesting		

Inheritance Graph

CommunityInvestmentFundContract

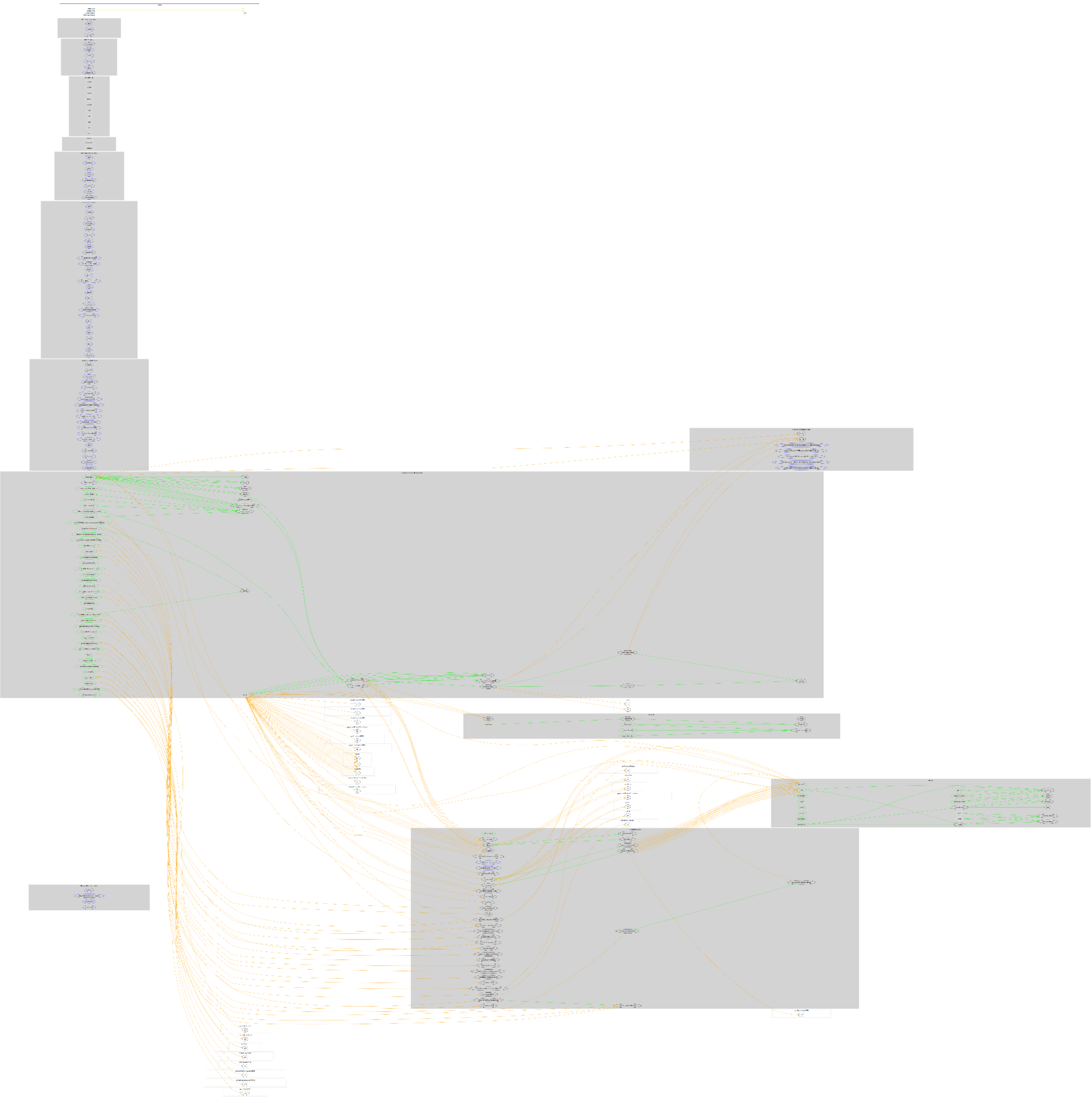


TokenVesting

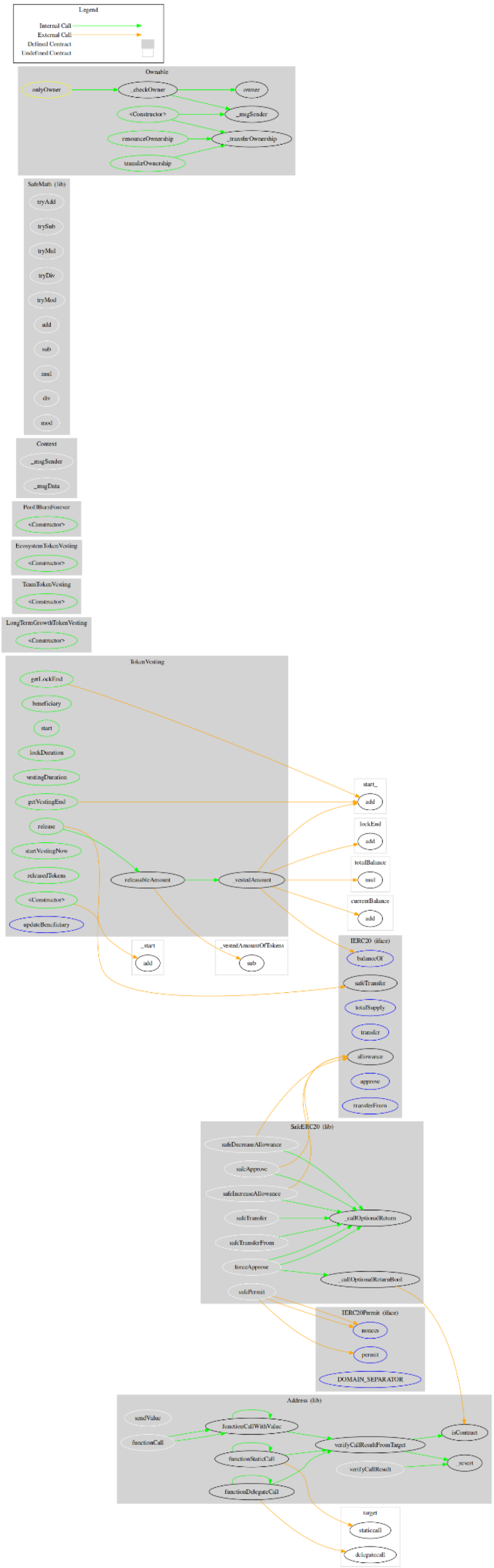


Flow Graph

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 team has acknowledged the issues.

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