Cyberscope

Audit Report

CoriBot

December 2023

SHA256 b40883a7931bb478929998b34685e994fa8badc6a15245d09d6cf2528c4b209d

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	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	OCTD	Transfers Contract's Tokens	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	CR	Code Repetition	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	CoribotV2
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Testing Deploy	https://mumbai.polygonscan.com/address/0x341d42aafaa75bc49977b49856031547add5a2e4
Address	0x341d42aafaa75bc49977b49856031547add5a2e4
Network	MATIC Testnet
Symbol	CORI
Decimals	18
Total Supply	10,000,000

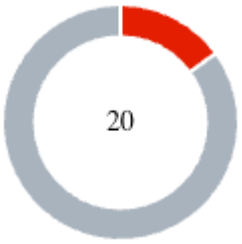
Audit Updates

Initial Audit	17 Dec 2023

Source Files

Filename	SHA256
contracts/BEP20Token.sol	b40883a7931bb478929998b34685e994fa8badc6a15245d09d6cf2528c4b209d

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/BEP20Token.sol#L1259
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 enabled the owner will not be able to disable them again.

```
if (!tradingActive) {
    require(
        _isExcludedFromFees[from] || _isExcludedFromFees[to],
        "Trading is not active."
    );
}
```

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.

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/BEP20Token.sol#L1459,1470
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling either the `blacklist` or the `blacklistLiquidityPool` function.

```
function blacklist(address _addr) public onlyOwner {
    require(!blacklistRenounced, "Team has revoked blacklist rights");
    require(
//            _addr != address(uniswapV2Pair) && _addr !=
address(0x10ED43C718714eb63d5aA57B78B54704E256024E), // TODO : mainnet
        _addr != address(uniswapV2Pair) && _addr !=
address(0xD99D1c33F9fC3444f8101754aBC46c52416550D1),
        "Cannot blacklist token's v2 router or v2 pool."
    );
    blacklisted[_addr] = true;
}

// @dev blacklist v3 pools; can unblacklist() down the road to suit
project and community
function blacklistLiquidityPool(address lpAddress) public onlyOwner {
    require(!blacklistRenounced, "Team has revoked blacklist rights");
    require(
//            lpAddress != address(uniswapV2Pair) && lpAddress !=
address(0x10ED43C718714eb63d5aA57B78B54704E256024E), // TODO : mainnet
        lpAddress != address(uniswapV2Pair) && lpAddress !=
address(0xD99D1c33F9fC3444f8101754aBC46c52416550D1),
        "Cannot blacklist token's v2 router or v2 pool."
    );
    blacklisted[lpAddress] = true;
}
```

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.

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.

OCTD - Transfers Contract's Tokens

Criticality	Critical
Location	contracts/BEP20Token.sol#L1435,1441
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling either the `withdrawStuckCoribot` or the `withdrawStuckToken` function.

The `tokensForLiquidity` , `tokensForTeam` , and `tokensForRevShare` accumulate tokens that are meant to be swapped. If the owner calls either of the aforementioned functions, then the contract will transfer all of its tokens to either the owner's address or the provided address. The contract's balance will become zero, but the contract does not reset the `tokensForLiquidity` , `tokensForTeam` , and `tokensForRevShare` variables back to zero. Afterward, when the contract swaps its tokens again, these variables will have an amount greater than the contract's balance actual amount. As a result, the transaction will revert.

```
function withdrawStuckCoribot() external onlyOwner {
    uint256 balance = IERC20(address(this)).balanceOf(address(this));
    IERC20(address(this)).transfer(msg.sender, balance);
    payable(msg.sender).transfer(address(this).balance);
}

function withdrawStuckToken(address _token, address _to) external onlyOwner {
    require(_token != address(0), "_token address cannot be 0");
    uint256 _contractBalance = IERC20(_token).balanceOf(address(this));
    IERC20(_token).transfer(_to, _contractBalance);
}
```

Recommendation

The team is strongly advised to keep these variables in sync with the contract's balance. Specifically, the contract should reset these variables back to zero within the `withdrawStuckCoribot` and `withdrawStuckToken` functions. 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.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L984
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 declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
address public constant deadAddress = address(0xdead);
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1329,1330,1331,1336,1337,1338
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.

```
tokensForLiquidity += (fees * sellLiquidityFee) / sellTotalFees;
tokensForTeam += (fees * sellTeamFee) / sellTotalFees;
tokensForRevShare += (fees * sellRevShareFee) / sellTotalFees;

tokensForLiquidity += (fees * buyLiquidityFee) / buyTotalFees;
tokensForTeam += (fees * buyTeamFee) / buyTotalFees;
tokensForRevShare += (fees * buyRevShareFee) / buyTotalFees;
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L
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.

```
function blacklist(address _addr) public onlyOwner {
    require(!blacklistRenounced, "Team has revoked blacklist rights");
    require(
//            _addr != address(uniswapV2Pair) && _addr !=
address(0x10ED43C718714eb63d5aA57B78B54704E256024E), // TODO : mainnet
        _addr != address(uniswapV2Pair) && _addr !=
address(0xD99D1c33F9fC3444f8101754aBC46c52416550D1),
        "Cannot blacklist token's v2 router or v2 pool."
    );
    blacklisted[_addr] = true;
}

// @dev blacklist v3 pools; can unblacklist() down the road to suit
project and community
function blacklistLiquidityPool(address lpAddress) public onlyOwner {
    require(!blacklistRenounced, "Team has revoked blacklist rights");
    require(
//            lpAddress != address(uniswapV2Pair) && lpAddress !=
address(0x10ED43C718714eb63d5aA57B78B54704E256024E), // TODO : mainnet
        lpAddress != address(uniswapV2Pair) && lpAddress !=
address(0xD99D1c33F9fC3444f8101754aBC46c52416550D1),
        "Cannot blacklist token's v2 router or v2 pool."
    );
    blacklisted[lpAddress] = true;
}
```

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.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1411,1413
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
totalTokensToSwap - (tokensForLiquidity / 2)
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1199
Status	Unresolved

Description

The `setAutomatedMarketMakerPair` function allows any user to set any arbitrary value without validation to the `automatedMarketMakerPairs` mapping, which is supposed to hold Uniswap pair addresses. This lack of validation can lead to unintended behavior, including the potential disruption of the contract's intended functionality.

```
function setAutomatedMarketMakerPair(address pair, bool value)
public
onlyOwner
{
    require(
        pair != uniswapV2Pair,
        "The pair cannot be removed from automatedMarketMakerPairs"
    );

    _setAutomatedMarketMakerPair(pair, value);
}

function _setAutomatedMarketMakerPair(address pair, bool value) private {
    automatedMarketMakerPairs[pair] = value;

    emit SetAutomatedMarketMakerPair(pair, value);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1028
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 declares certain events in its code. However, these events 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 UpdateUniswapV2Router(
    address indexed newAddress,
    address indexed oldAddress
);
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1114,1120,1162,1167,1456,1466,1477,1482
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.

```
tradingActive = true;
limitsInEffect = false;
_isExcludedMaxTransactionAmount[updAds] = isEx;
swapEnabled = enabled;
blacklistRenounced = true;
...
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1114,1120,1162,1167,1195,1456,1466,1477,1482
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
tradingActive = true;
limitsInEffect = false;
_isExcludedMaxTransactionAmount[updAds] = isEx;
swapEnabled = enabled;
_isExcludedFromFees[account] = excluded;
blacklistRenounced = true;
...
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/BEP20Token.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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1451
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(success)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L841,843,874,920,1022,1037,1042,1171,1172,1173,1183,1184,1185,1441,1459,1481
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);
mapping(address => bool) public _isExcludedMaxTransactionAmount

event revShareWalletUpdated(
...
    );

event teamWalletUpdated(
        address indexed newWallet,
        address indexed oldWallet
    );
uint256 _revShareFee

...
```

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/BEP20Token.sol#L1138,1147,1155,1175,1187
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.

```
swapTokensAtAmount = newAmount
maxTransactionAmount = newNum * (10**18)
maxWallet = newNum * (10**18)
buyRevShareFee = _revShareFee
sellRevShareFee = _revShareFee
```

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	contracts/BEP20Token.sol#L472
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 _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
    }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), 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	contracts/BEP20Token.sol#L1328,1329,1330,1331,1335,1336,1337,1338
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.

```
fees = amount.mul(sellTotalFees).div(100)
tokensForLiquidity += (fees * sellLiquidityFee) / sellTotalFees
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1075
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 totalSupply = 10_000_000 * 1e18
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1219,1224,1448
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.

```
revShareWallet = newRevShareWallet
teamWallet = newWallet

(bool success, ) = toAddr.call{
        value: address(this).balance
        } ("")
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/BEP20Token.sol#L1437,1444
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.

```
IERC20(address(this)).transfer(msg.sender, balance)
IERC20(_token).transfer(_to, _contractBalance)
```

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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IPancakeFactory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-

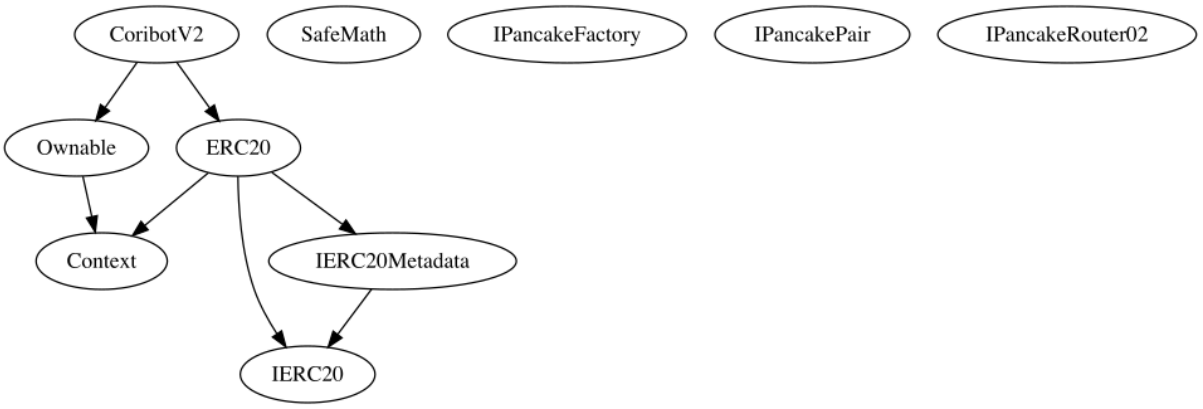
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IPancakePair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-

	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IPancakeRouter02	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
CoribotV2	Implementation	ERC20, Ownable		

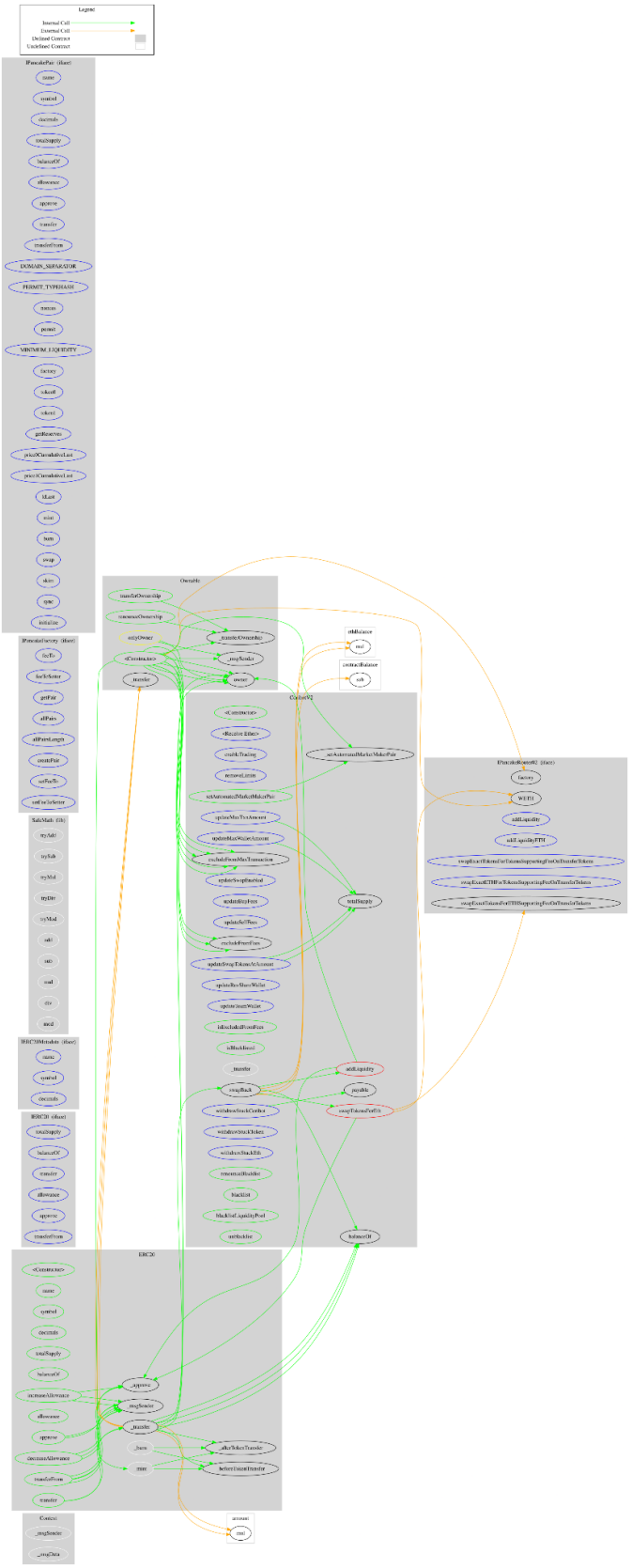
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateRevShareWallet	External	✓	onlyOwner
	updateTeamWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	isBlacklisted	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	withdrawStuckCoribot	External	✓	onlyOwner

	withdrawStuckToken	External	✓	onlyOwner
	withdrawStuckEth	External	✓	onlyOwner
	renounceBlacklist	Public	✓	onlyOwner
	blacklist	Public	✓	onlyOwner
	blacklistLiquidityPool	Public	✓	onlyOwner
	unblacklist	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IPancakePair (iface)
name
symbol
decimals
totalSupply
balanceOf
allowance
approve
transfer
transferFrom
DOMAIN_SEPARATOR
PERMIT_TYPEHASH
nonces
permit
MINIMUM_LIQUIDITY
factory
token0
token1
getReserves
price0CumulativeLast
price1CumulativeLast
kLast
mint
burn
swap
skim
sync
initialize
IPancakeFactory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
IBEP20Metadata (iface)
name
symbol
decimals
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
transferOwnership
renounceOwnership
_transferOwnership
onlyOwner
_msgSender
<Constructor>
owner
ethBalance
contractBalance
CoriboV2
<Constructor>
<Receive Ether>
enableTrading
removeLimits
setAutomatedMarketMakerPair
updateMaxTxAmount
updateMaxWalletAmount
excludeFromMaxTransaction
updateSwapEnabled
updateBuyFees
updateSellFees
excludeFromFees
updateSwapTokensAtAmount
updateBuyBurnWallet
updateTeamWallet
isExcludedFromFees
isBlacklisted
_transfer
swapBack
withdrawStuckCoribot
withdrawStuckToken
withdrawStuckBNB
removeFromBlacklist
blacklist
blacklistLiquidityPool
setBlacklist
totalSupply
addLiquidity
payable
swapTokensForEth
balanceOf
IPancakeRouter02 (iface)
factory
WETH
addLiquidity
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
increaseAllowance
allowance
approve
decreaseAllowance
transferFrom
transfer
_approve
_msgSender
_transfer
_burn
mint
_afterTokenTransfer
_beforeTokenTransfer
Context
_msgSender
_msgData
account
mint

Summary

CoriBot 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 massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 5% and 10% sell fees.

Disclaimer

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

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

About Cyberscope

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

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

The Cyberscope team

https://www.cyberscope.io