Cyberscope

Audit Report

Karma

July 2023

Network	BSC
Address	0xBC570B260c52A032DE9eF06087e91b2C5104C76B
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ZD	Zero Division	Unresolved
●	MC	Missing Check	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	DDP	Decimal Division Precision	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
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved

●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	Karma
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0xbc570b260c52a032de9ef06087e91b2c5104c76b
Address	0xbc570b260c52a032de9ef06087e91b2c5104c76b
Network	BSC
Symbol	KARMA
Decimals	18
Total Supply	4,444,000,000

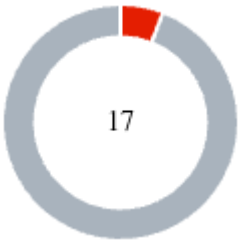
Audit Updates

Initial Audit	06 Jul 2023

Source Files

Filename	SHA256
Karma.sol	419a562decf373bc52da8ba7a7535b12ef6a047538df4ed82be8afee616f842a

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	16	0	0	0

ZD - Zero Division

Criticality	Critical
Location	Karma.sol#L537,610
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

If all fees are set to zero then `_totalFee` will be zero. As a result, the transaction will revert due to zero division.

```
uint256 proportionReflected = buying == true
    ? proportionFeeAmount.mul(reflectionFeeBuy).div(totalFeeBuy)
    : proportionFeeAmount.mul(reflectionFeeSell).div(totalFeeSell);

uint256 totalBNBFee = _totalFee;
uint256 amountBNBMarketing = amountBNB.mul(marketingFeeSell).div(
    totalBNBFee
);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

MC - Missing Check

Criticality	Minor / Informative
Location	Karma.sol#L631
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.

The variable `swapThreshold` sets a threshold where the contract will trigger the swap functionality. If the swapThreshold is set to zero, the contract will perform redundant swaps.

```
function setSwapBackSettings(
    bool _enabled,
    uint256 _amountS,
    uint256 _amountL,
    bool _alternate
) external onlyOwner {
    alternateSwaps = _alternate;
    claimingFees = _enabled;
    smallSwapThreshold = _amountS;
    largeSwapThreshold = _amountL;
    swapThreshold = smallSwapThreshold;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	Karma.sol#L313,314
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
router = IDEXRouter(ROUTER);
pair = IDEXFactory(router.factory()).createPair(
    address(this),
    router.WETH()
);
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Karma.sol#L645,675,681,685,692,699,700
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.

```
tradingOpen = true;
isFeeExempt[holder] = exempt;
feeExemptSetter = _feeExemptSetter;
feeExemptSetter = address(0);
isTxLimitExempt[holder] = exempt;
marketingFeeReceiver = _marketingFeeReceiver;
teamFeeReceiver = _teamFeeReceiver;
```

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.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	Karma.sol#L639,640
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.

The `largeSwapThreshold` should be greater than the `smallSwapThreshold`.

```
smallSwapThreshold = _amountS;
largeSwapThreshold = _amountL;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Karma.sol#L645,675,681,692,699,700
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 modifies the state of certain variables even when their current state matches the provided argument. As a result, the contract performs redundant storage writes.

```
tradingOpen = true;
isFeeExempt[holder] = exempt;
feeExemptSetter = _feeExemptSetter;
isTxLimitExempt[holder] = exempt;
marketingFeeReceiver = _marketingFeeReceiver;
teamFeeReceiver = _teamFeeReceiver;
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Karma.sol#L631
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variables `smallSwapThreshold` and `largeSwapThreshold` set a threshold where the contract will trigger the swap functionality. If the variables are 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 setSwapBackSettings(
    bool _enabled,
    uint256 _amountS,
    uint256 _amountL,
    bool _alternate
) external onlyOwner {
    alternateSwaps = _alternate;
    claimingFees = _enabled;
    smallSwapThreshold = _amountS;
    largeSwapThreshold = _amountL;
    swapThreshold = smallSwapThreshold;
}
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Karma.sol#L610,613
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.

```
uint256 totalBNBFee = _totalFee;
uint256 amountBNBMarketing = amountBNB.mul(marketingFeeSell).div(
    totalBNBFee
);
uint256 amountBNBteam = amountBNB.mul(teamFeeSell).div(totalBNBFee);
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Karma.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	Karma.sol#L313,314
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.

```
router
pair
```

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	Karma.sol#L260,279
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 _totalSupply = 4444 * 10 ** 6 * 10 ** _decimals
uint256 feeDenominator = 100
```

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	Karma.sol#L168,256,257,258,260,263,264,266,632,633,634,635,649,650,651,652,653,654,679,696,697
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 WETH() external pure returns (address);
string _name = "Karma"
string _symbol = "KARMA"
uint8 constant _decimals = 18
uint256 _totalSupply = 4444 * 10 ** 6 * 10 ** _decimals
mapping(address => uint256) public _rOwned
uint256 public _totalProportion = _totalSupply
mapping(address => mapping(address => uint256)) _allowances
bool _enabled
uint256 _amountS
uint256 _amountL
bool _alternate
uint256 _reflectionFeeBuy
uint256 _marketingFeeBuy

...
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	Karma.sol#L681
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
feeExemptSetter = _feeExemptSetter
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Karma.sol#L639,656
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.

```
smallSwapThreshold = _amountS
reflectionFeeBuy = _reflectionFeeBuy
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Karma.sol#L528,537
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
uint256 proportionFeeAmount = buying == true
            ? proportionAmount.mul(getTotalFeeBuy(receiver == pair)).div(
                feeDenominator
            )
            : proportionAmount.mul(getTotalFeeSell(receiver == pair)).div(
                feeDenominator
            )

uint256 proportionReflected = buying == true
            ? proportionFeeAmount.mul(reflectionFeeBuy).div(totalFeeBuy)
            : proportionFeeAmount.mul(reflectionFeeSell).div(totalFeeSell)
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Karma.sol#L528,537
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 proportionFeeAmount = buying == true
            ? proportionAmount.mul(getTotalFeeBuy(receiver == pair)).div(
                feeDenominator
            )
            : proportionAmount.mul(getTotalFeeSell(receiver == pair)).div(
                feeDenominator
            )
uint256 proportionReflected = buying == true
            ? proportionFeeAmount.mul(reflectionFeeBuy).div(totalFeeBuy)
            : proportionFeeAmount.mul(reflectionFeeSell).div(totalFeeSell)
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Karma.sol#L681,699,700
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.

```
feeExemptSetter = _feeExemptSetter
marketingFeeReceiver = _marketingFeeReceiver
teamFeeReceiver = _teamFeeReceiver
```

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.

Functions Analysis

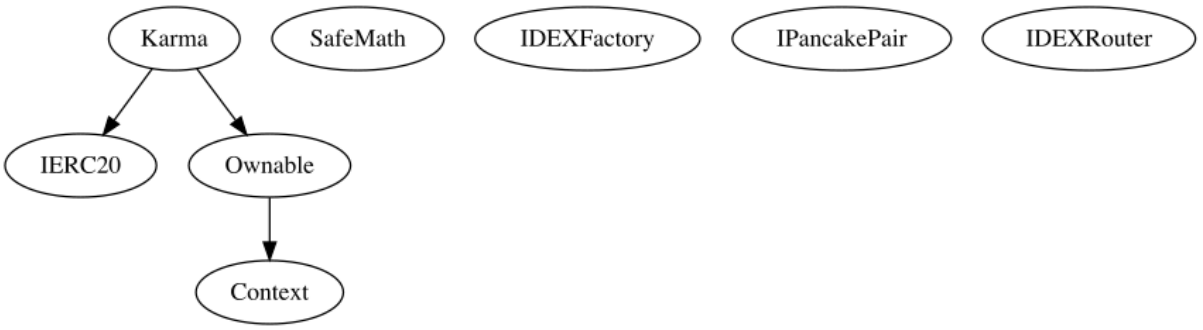
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
Context	Implementation			
	_msgSender	Internal		

	_msgData	Internal		
IDEXFactory	Interface			
	createPair	External	✓	-
IPancakePair	Interface			
	sync	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
Karma	Implementation	IERC20, Ownable		
		Public	✓	-
		External	Payable	-

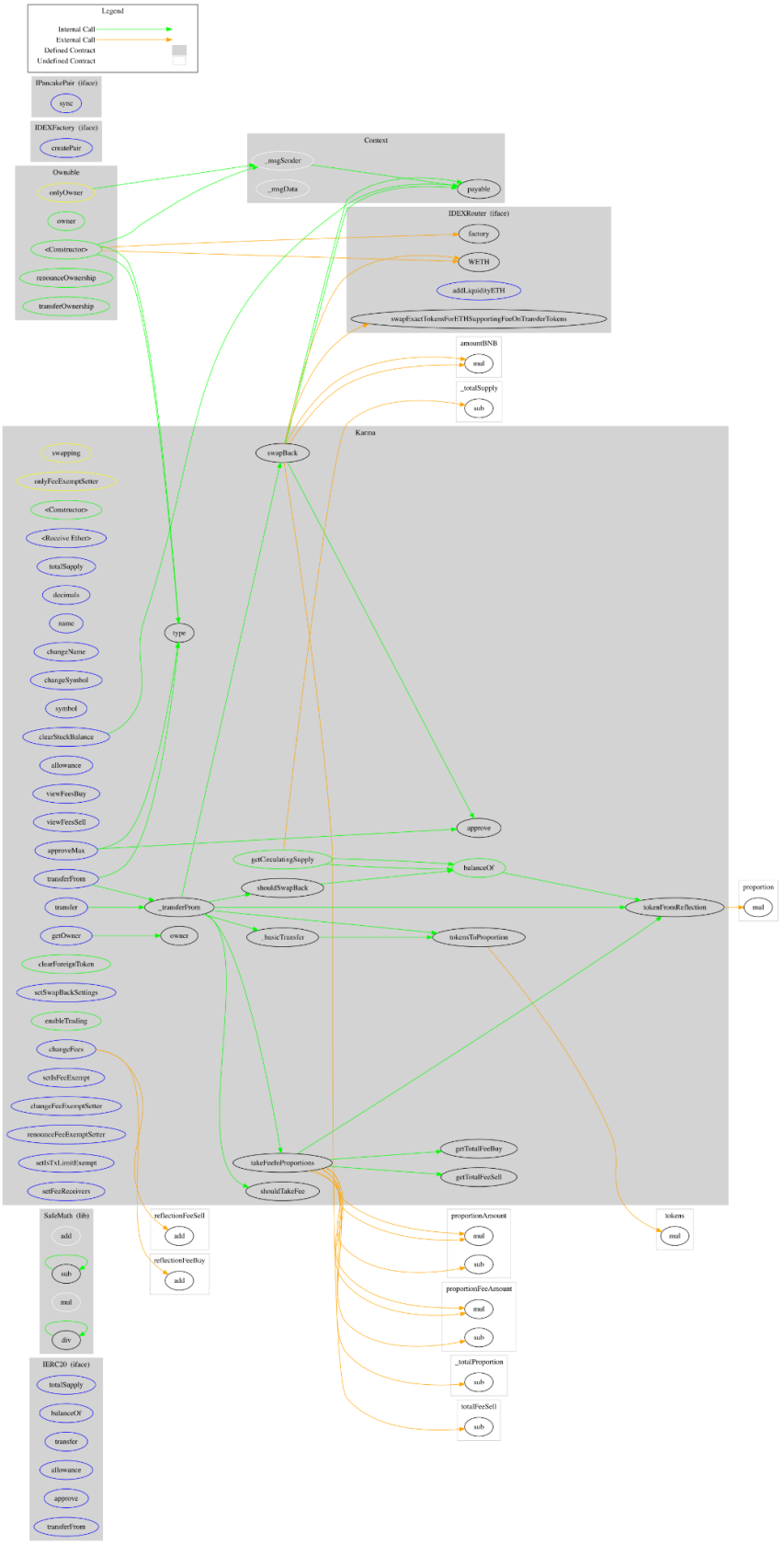
	totalSupply	External		-
	decimals	External		-
	name	External		-
	changeName	External	✓	onlyOwner
	changeSymbol	External	✓	onlyOwner
	symbol	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	viewFeesBuy	External		-
	viewFeesSell	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	tokensToProportion	Public		-
	tokenFromReflection	Public		-
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	getTotalFeeBuy	Public		-
	getTotalFeeSell	Public		-
	takeFeeInProportions	Internal	✓	

	clearStuckBalance	External	✓	onlyOwner
	clearForeignToken	Public	✓	-
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	setSwapBackSettings	External	✓	onlyOwner
	enableTrading	Public	✓	onlyOwner
	changeFees	External	✓	onlyOwner
	setIsFeeExempt	External	✓	onlyFeeExempt Setter
	changeFeeExemptSetter	External	✓	onlyFeeExempt Setter
	renounceFeeExemptSetter	External	✓	onlyFeeExempt Setter
	setIsTxLimitExempt	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	getCirculatingSupply	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IPancakePair (iface)
sync
IDEXFactory (iface)
createPair
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
transferOwnership
Context
_msgSender
_msgData
payable
IDEXRouter (iface)
factory
WETH
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens
amountBNB
mul
_totalSupply
sub
Karma
swapping
onlyFeeExemptSetter
<Constructor>
<Receive Ether>
totalSupply
decimals
name
changeName
changeSymbol
symbol
clearStuckBalance
allowance
viewFeesBuy
viewFeesSell
approveMax
transferFrom
transfer
getOwner
clearForeignToken
setSwapBackSettings
enableTrading
changeFees
setIsFeeExempt
changeFeeExemptSetter
renounceFeeExemptSetter
setIsTxLimitExempt
setFeeReceivers
type
swapBack
approve
getCirculatingSupply
balanceOf
shouldSwapBack
_transferFrom
owner
basicTransfer
takeFeeInProportions
shouldTakeFee
tokensToProportion
tokenFromReflection
proportion
mul
getTotalFeeBuy
getTotalFeeSell
SafeMath (lib)
add
sub
mul
div
reflectionFeeSell
reflectionFeeBuy
proportionAmount
proportionFeeAmount
_totalProportion
totalFeeSell
tokens
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Karma contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Karma 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 10% 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