Cyberscope

Audit Report

The Kingdomcoin

May 2023

Network	BSC
Address	0x06Dc293c250e2fB2416A4276d291803fc74fb9B5
Audited by	© cyberscope

Table of Contents

Table of Contents 1

Review 2

Audit Updates 2

Source Files 2

Findings Breakdown 3

Analysis 4

ELFM - Exceeds Fees Limit 5

Description 5

Recommendation 5

BC - Blacklists Addresses 6

Description 6

Recommendation 6

Diagnostics 7

ZD - Zero Division 8

Description 8

Recommendation 8

CO - Code Optimization 9

Description 9

Recommendation 9

PVC - Price Volatility Concern 11

Description 11

Recommendation 11

RSML - Redundant SafeMath Library 12

Description 12

Recommendation 12

IDI - Immutable Declaration Improvement 13

Description 13

Recommendation 13

L04 - Conformance to Solidity Naming Conventions 14

Description 14

Recommendation 14

L05 - Unused State Variable 15

Description 15

Recommendation 15

L09 - Dead Code Elimination 16

Description 16

Recommendation 16

L14 - Uninitialized Variables in Local Scope 18

Description 18

Recommendation 18
L16 - Validate Variable Setters 19
Description 19
Recommendation 19
L17 - Usage of Solidity Assembly 20
Description 20
Recommendation 20
L22 - Potential Locked Ether 21
Description 21
Recommendation 21
Functions Analysis 22
Inheritance Graph 27
Flow Graph 28
Summary 29
Disclaimer 30
About Cyberscope 31

Review

Contract Name	KingdomCoin
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x06dc293c250e2fb2416a4276d291803fc74fb9b5
Address	0x06dc293c250e2fb2416a4276d291803fc74fb9b5
Network	BSC
Symbol	TKC
Decimals	18
Total Supply	240.000.000

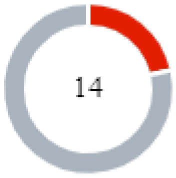
Audit Updates

Initial Audit	08 May 2023

Source Files

Filename	SHA256
KingdomCoin.sol	7884a5df4f50c282335d386836ca04832591703c3d9199cea3404ee7289263c6

Findings Breakdown



●	Critical	3
●	Medium	0
●	Minor / Informative	11

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	3	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	11	0	0	0

Analysis

🔴 Critical 🟠 Medium ⚪ Minor / Informative 🔵 Pass

Severity	Code	Description	Status
🔵	ST	Stops Transactions	Passed
🔵	OCTD	Transfers Contract's Tokens	Passed
🔵	OTUT	Transfers User's Tokens	Passed
🔴	ELFM	Exceeds Fees Limit	Unresolved
🔵	ULTW	Transfers Liquidity to Team Wallet	Passed
🔵	MT	Mints Tokens	Passed
🔵	BT	Burns Tokens	Passed
🔴	BC	Blacklists Addresses	Unresolved

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	KingdomCoin.sol#L873
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `updateFees` function with a high percentage value.

```
function updateFees(uint256 _buyBack, uint256 operationPerc)
external onlyOwner {
  require(_buyBack.add(operationPerc) <= maximumTotalFee, "Fees
exceed maximum");
  emit FeesUpdated(operationPerc, _buyBack);
  operationalFee = operationPerc;
  buyBackFee = _buyBack;
  totalFees = operationalFee.add(buyBackFee);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. 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.

BC - Blacklists Addresses

Criticality	Critical
Location	KingdomCoin.sol#L824
Status	Unresolved

Description

The contract owner has the authority to massively stop addresses from transactions. The owner may take advantage of it by calling the `blackList` function.

```
function blackList(address[] memory black, bool status)
external onlyOwner {
  uint256 lent = black.length;
  for (uint256 i; i < lent; ) {
        _isBlacklist[black[i]] = status;
        emit BlackList(black[i]);
        unchecked {
            i++;
        }
  }
}
```

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 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.

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🔴	ZD	Zero Division	Unresolved
●	CO	Code Optimization	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L22	Potential Locked Ether	Unresolved

ZD - Zero Division

Criticality	Critical
Location	KingdomCoin.sol#L974
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.

The variable `totalFees` can be set to zero, and as a result, the transfer transaction will revert.

```
function swapAndSendDividends(uint256 tokenAmount) private {
    // transfer buyBack tokens to the designated wallet
    uint256 subBuyBackFee =
tokenAmount.mul(buyBackFee).div(totalFees);
```

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.

CO - Code Optimization

Criticality	Minor / Informative
Location	KingdomCoin.sol#L739,810,750
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 variable `maxSellTransactionAmount` and the mapping `canTransferBeforeTradingIsEnabled` are not utilized in the contract's implemantaion. Hence, they are redundant.

```
uint256 public maxSellTransactionAmount = 10_000_000 *
(10**18);

function updateMaximumSales(uint256 newMaximum) external
onlyOwner {
    require(newMaximum > totalSupply().div(5), "Swap amount
cannot be lesser than 5% of total supply.");
    emit UpdateMaximumSell(maxSellTransactionAmount);
    maxSellTransactionAmount = newMaximum;
}

mapping (address => bool) private
canTransferBeforeTradingIsEnabled;
```

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.

The variable `maxSellTransactionAmount`, the method `updateMaximumSales`, and the mapping `canTransferBeforeTradingIsEnabled` could be removed from the contract implementation.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	KingdomCoin.sol#L804
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` 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 updateSwapTokensAtAmount(uint256 newAmount) external
onlyOwner {
    require(newAmount < totalSupply(), "Swap amount cannot be
higher than total supply.");
    swapTokensAtAmount = newAmount;
    emit SwapTokenAtAmount(swapTokensAtAmount);
}
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	KingdomCoin.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	KingdomCoin.sol#L791
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
tradingEnabled
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	KingdomCoin.sol#L685,869
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);
uint256 _buyBack
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	KingdomCoin.sol#L188
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	KingdomCoin.sol#L234,240,251,626
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function toUint256Safe(int256 a) internal pure returns
(uint256) {
...
    }

function toInt256Safe(uint256 a) internal pure returns (int256)
{
    int256 b = int256(a);
    require(b >= 0);
    return b;
  }

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	KingdomCoin.sol#L822,846
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 i
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	KingdomCoin.sol#L864,865
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.

```
operationsWallet = newOperationalWallet
buyBackWallet = newBuyBackWallet
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	KingdomCoin.sol#L884
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { codehash := extcodehash(account) }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	KingdomCoin.sol#L797
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {}
```

Recommendation

The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

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
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

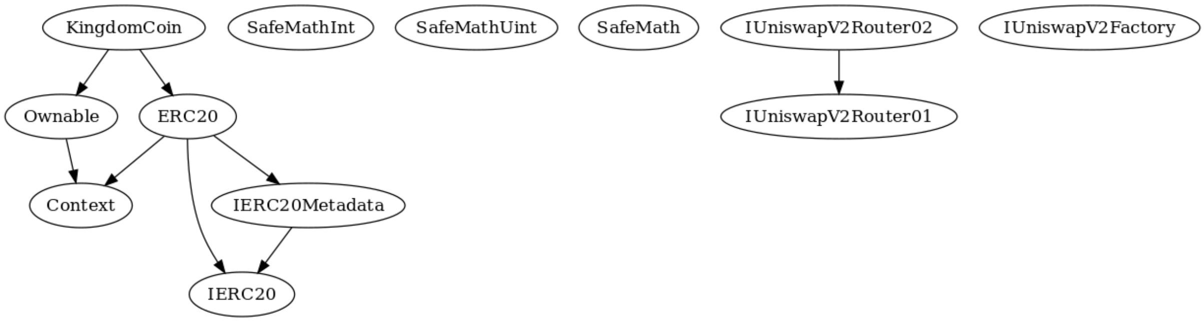
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

	mod	Internal		
	mod	Internal		
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	✓	

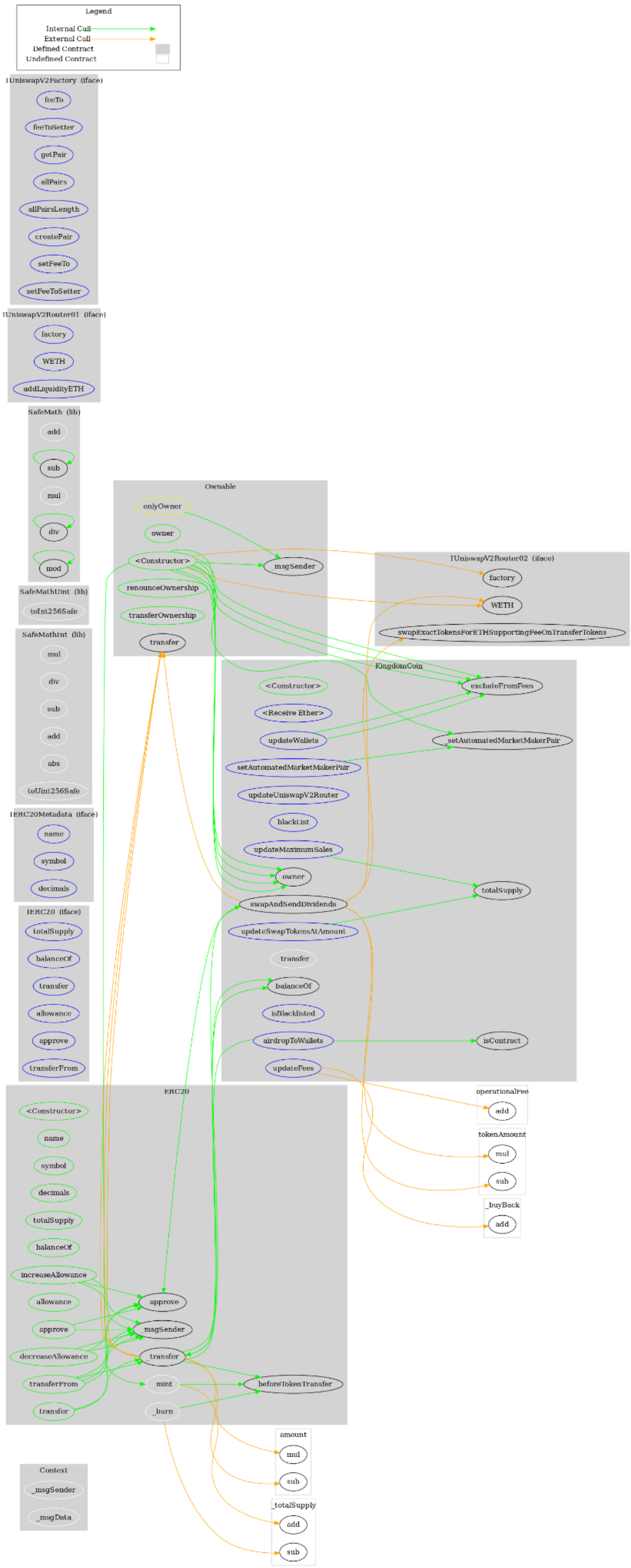
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
KingdomCoin	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateSwapTokensAtAmount	External	✓	onlyOwner

	updateMaximumSales	External	✓	onlyOwner
	updateUniswapV2Router	External	✓	onlyOwner
	blackList	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	airdropToWallets	External	✓	onlyOwner
	updateWallets	External	✓	onlyOwner
	updateFees	External	✓	onlyOwner
	isContract	Internal		
	_setAutomatedMarketMakerPair	Private	✓	
	_transfer	Internal	✓	
	swapAndSendDividends	Private	✓	
	isBlacklisted	External		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
IUniswapV2Router01 (iface)
factory
WETH
addLiquidityETH
SafeMath (lib)
add
sub
mul
div
mod
SafeMathUint (lib)
toInt256Safe
SafeMathInt (lib)
mul
div
sub
add
abs
toUint256Safe
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
owner
<Constructor>
msgSender
renounceOwnership
transferOwnership
transfer
IUniswapV2Router02 (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
KingdomCoin
<Constructor>
<Receive Ether>
updateWallets
setAutomatedMarketMakerPair
updateUniswapV2Router
blackList
updateMaximumSales
owner
swapAndSendDividends
updateSwapTokensAtAmount
transfer
balanceOf
isBlacklisted
airdropToWallets
updateFees
excludeFromFees
_setAutomatedMarketMakerPair
totalSupply
isContract
operationalFee
add
tokenAmount
mul
sub
_buyBack
add
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
increaseAllowance
allowance
approve
decreaseAllowance
transferFrom
transfer
approve
msgSender
transfer
mint
_burn
beforeTokenTransfer
amount
mul
sub
_totalSupply
add
sub
Context
_msgSender
_msgData

Summary

The Kingdomcoin 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 manipulating the fees 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.

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