



Cyberscope

Audit Report

Random Coin

July 2023

Network ETH

Address 0x16d782383Cdf7Be25673b56715a179894476dAcA

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
●	EPC	Existing Pair Creation	Unresolved
●	RFM	Redundant Fee Mechanism	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	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

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
ST - Stops Transactions	7
Description	7
Recommendation	7
BC - Blacklists Addresses	8
Description	8
Recommendation	8
EPC - Existing Pair Creation	9
Description	9
Recommendation	10
RFM - Redundant Fee Mechanism	11
Description	11
Recommendation	12
AOI - Arithmetic Operations Inconsistency	13
Description	13
Recommendation	13
FSA - Fixed Swap Address	14
Description	14
Recommendation	14
ULTW - Transfers Liquidity to Team Wallet	15
Description	15
Recommendation	15
RSML - Redundant SafeMath Library	17
Description	17
Recommendation	17
IDI - Immutable Declaration Improvement	18
Description	18
Recommendation	18
L02 - State Variables could be Declared Constant	19
Description	19
Recommendation	19
L04 - Conformance to Solidity Naming Conventions	20
Description	20

Recommendation	21
Functions Analysis	22
Inheritance Graph	25
Flow Graph	26
Summary	27
Disclaimer	28
About Cyberscope	29

Review

Contract Name	RANDOM
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://etherscan.io/address/0x16d782383cdf7be25673b56715a179894476daca
Address	0x16d782383cdf7be25673b56715a179894476daca
Network	ETH
Symbol	RANDOM
Decimals	9
Total Supply	1,000,000

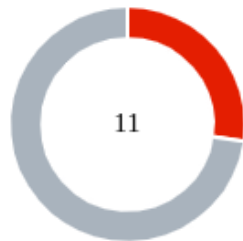
Audit Updates

Initial Audit	10 Jul 2023
---------------	-------------

Source Files

Filename	SHA256
RANDOM.sol	379f79ee0bbb32e75179cda14e76e0d7195f6ac3a9eccef86bd0a35ab9120898

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

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

```
function openTrading() external onlyOwner() {
    require(!tradingOpen, "trading is already open");
    uniswapV2Router =
    IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    _approve(address(this), address(uniswapV2Router), _tTotal);
    uniswapV2Pair =
    IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(t
his), uniswapV2Router.WETH());
    uniswapV2Router.addLiquidityETH({value:
address(this).balance} (address(this), balanceOf(address(this)), 0, 0
, owner(), block.timestamp);
    IERC20(uniswapV2Pair).approve(address(uniswapV2Router),
type(uint).max);
    swapEnabled = true;
    tradingOpen = 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. 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	RANDOM.sol#L297
Status	Unresolved

Description

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

```
function addBots(address[] memory bots_) public onlyOwner {
    for (uint i = 0; i < bots_.length; i++) {
        bots[bots_[i]] = 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. 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.

EPC - Existing Pair Creation

Criticality	Critical
Location	RANDOM.sol#L317
Status	Unresolved

Description

The contract is using the `openTrading` function to open trading and creating a new pair on the exchange, approving the router for the total supply of tokens, and adding liquidity.

However, the function does not handle the scenario where a pair already exists prior to its execution.

If a pair is created before the `openTrading` function is called, the contract will not be able to execute the `openTrading` function and will be stuck. This is because the `createPair` function in the Uniswap V2 Factory contract returns the address of the already created pair if it exists, instead of creating a new one. This behavior can lead to a situation where the `openTrading` function cannot be executed if a pair is already created, which can potentially block trading.

```
function openTrading() external onlyOwner() {
    require(!tradingOpen, "trading is already open");
    uniswapV2Router =
    IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    _approve(address(this), address(uniswapV2Router), _tTotal);
    uniswapV2Pair =
    IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(this),
    uniswapV2Router.WETH());
    uniswapV2Router.addLiquidityETH({value:
    address(this).balance} (address(this), balanceOf(address(this)), 0, 0, owner(
    ), block.timestamp);
    IERC20(uniswapV2Pair).approve(address(uniswapV2Router),
    type(uint).max);
    swapEnabled = true;
    tradingOpen = true;
}
```

Recommendation

It is recommended to add a check to the `openTrading` function to verify if a pair already exists. This can be done by calling the `getPair` function of the Uniswap V2 Factory contract. If the `getPair` function returns the zero address, it means that the pair does not exist and the `createPair` function can be safely called. This will prevent the contract from getting stuck in case a pair is already created before the `openTrading` function is called.

RFM - Redundant Fee Mechanism

Criticality	Minor / Informative
Location	RANDOM.sol#L130,225,245
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 variables and functions as part of a fee mechanism. However, this mechanism is not being used by the transaction flow. As a result, these variables and functions are redundant.

the variables `_finalBuyTax`, `_initialBuyTax`, `_finalSellTax` and `_initialSellTax` are initialized to zero and remain unchanged throughout the contract. These variables are used in the calculation of `taxAmount` and will always evaluate to zero, regardless of the values of `_buyCount`, `_reduceBuyTaxAt` and `_reduceSellTaxAt`. This means that the fee mechanism, will not generate any fees.

```
uint256 private _initialBuyTax=0;
uint256 private _initialSellTax=0;
uint256 private _finalBuyTax=0;
uint256 private _finalSellTax=0;
...
taxAmount =
amount.mul(( _buyCount>_reduceBuyTaxAt)?_finalBuyTax:_initialBuyTax).div(100);

...
if(to == uniswapV2Pair && from!= address(this) ){
    taxAmount =
amount.mul(( _buyCount>_reduceSellTaxAt)?_finalSellTax:_initialSellTax).div(100);
}
```

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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	RANDOM.sol#L240,259
Status	Unresolved

Description

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

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

```
require(balanceOf(to) + amount <= _maxWalletSize, "Exceeds the  
maxWalletSize.");  
  
_balances[address(this)] = _balances[address(this)].add(taxAmount  
);
```

Recommendation

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

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	RANDOM.sol#L314
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.

```
function openTrading() external onlyOwner() {
    ...
    uniswapV2Router =
    IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    _approve(address(this), address(uniswapV2Router), _tTotal);
    uniswapV2Pair =
    IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(t
    his), uniswapV2Router.WETH());
    uniswapV2Router.addLiquidityETH{value:
    address(this).balance}(address(this),balanceOf(address(this)),0,0
    ,owner(),block.timestamp);
    ...
}
```

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.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	RANDOM.sol#L334
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the `_taxWallet` wallet. These funds have been accumulated from fees collected from the contract. The `_taxWallet` address may take advantage of it by calling the `manualSwap` method.

```
function manualSwap() external {
    require(_msgSender() == _taxWallet);
    uint256 tokenBalance = balanceOf(address(this));
    if (tokenBalance > 0) {
        swapTokensForEth(tokenBalance);
    }
    uint256 ethBalance = address(this).balance;
    if (ethBalance > 0) {
        sendETHToFee(ethBalance);
    }
}

function sendETHToFee(uint256 amount) private {
    _taxWallet.transfer(amount);
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. The team should carefully manage the private keys of the `_taxWallet`'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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	RANDOM.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	RANDOM.sol#L162
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.

```
_taxWallet
```

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	RANDOM.sol#L130,131,134,135,136,145,146
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 private _initialBuyTax=0
uint256 private _initialSellTax=0
uint256 private _reduceBuyTaxAt=0
uint256 private _reduceSellTaxAt=0
uint256 private _preventSwapBefore=20
uint256 public _taxSwapThreshold= 20000 * 10**_decimals
uint256 public _maxTaxSwap= 10000 * 10**_decimals
```

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	RANDOM.sol#L109,139,140,141,142,143,144,145,146,325
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);
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 1000000 * 10**_decimals
string private constant _name = unicode"RANDOM COIN"
string private constant _symbol = unicode"RANDOM"
uint256 public _maxTxAmount = 20000 * 10**_decimals
uint256 public _maxWalletSize = 20000 * 10**_decimals
uint256 public _taxSwapThreshold= 20000 * 10**_decimals
uint256 public _maxTaxSwap= 10000 * 10**_decimals
uint256 _newFee
```

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>.

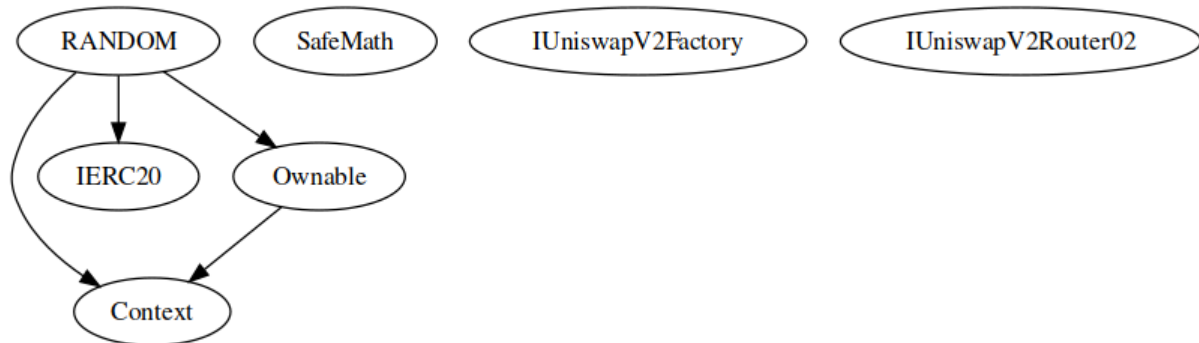
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
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		

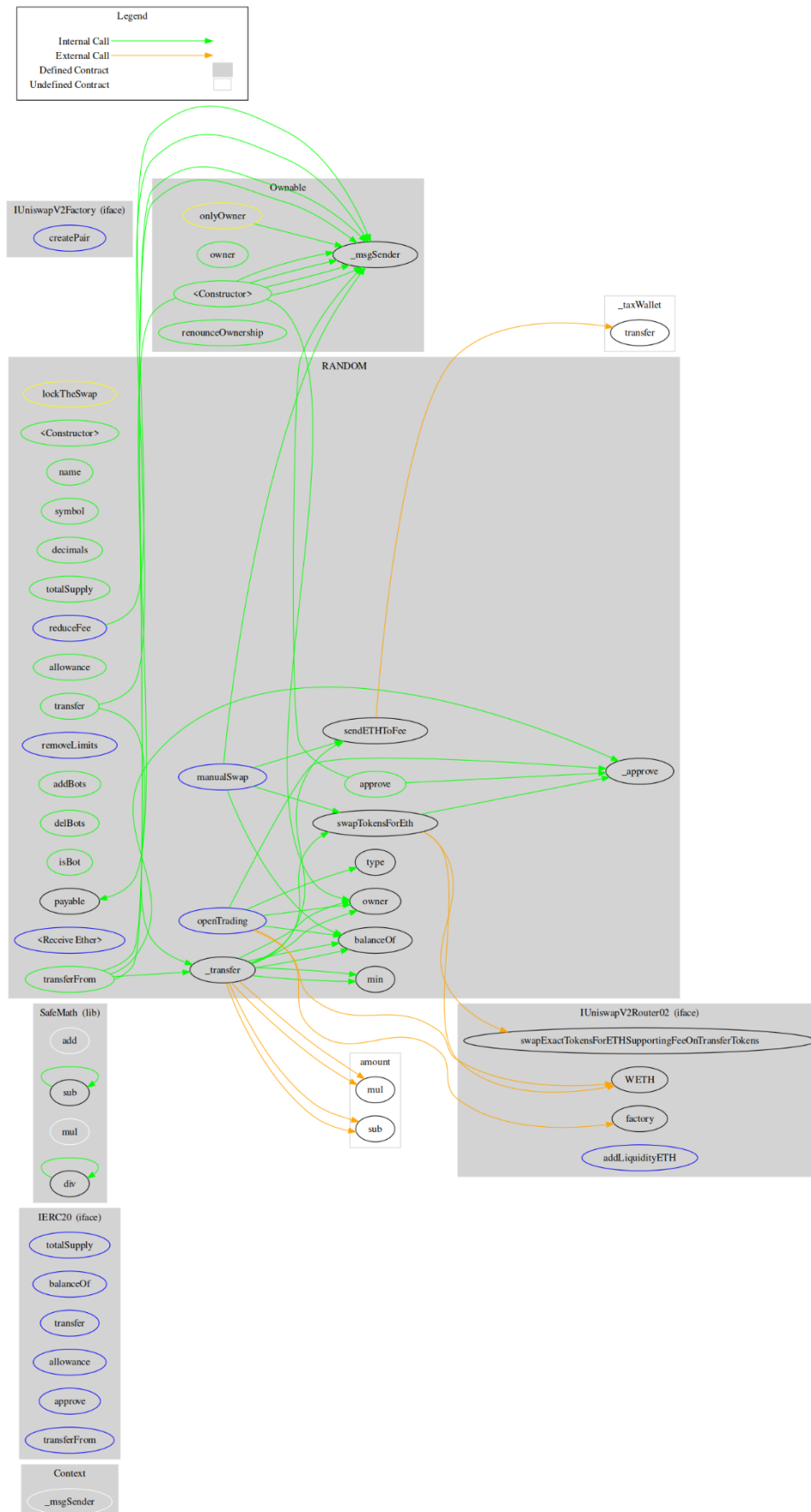
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
RANDOM	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-

	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	removeLimits	External	✓	onlyOwner
	sendETHToFee	Private	✓	
	addBots	Public	✓	onlyOwner
	delBots	Public	✓	onlyOwner
	isBot	Public		-
	openTrading	External	✓	onlyOwner
	reduceFee	External	✓	-
		External	Payable	-
	manualSwap	External	✓	-

Inheritance Graph



Flow Graph



Summary

Random Coin 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 will eliminate all the contract threats. There is also a fee mechanism in the contract, but it's ineffective as the tax variables are statically set to zero, resulting in zero 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>