Cyberscope

Audit Report

ReusedCoin

March 2024

SHA256	c7782b71868a6fea6533624318418d341cc16335fd736cfc3aac3c55dde9b480
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	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	FRV	Fee Restoration Vulnerability	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Review

Contract Name	ReusedCoin
Testing Deploy	https://mumbai.polygonscan.com/address/0xe84945d0fc45807dc12a49b79dbdd8cbf44f7268
Badge Eligibility	Yes

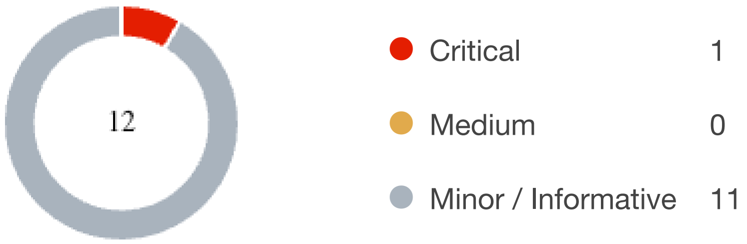
Audit Updates

Initial Audit	15 Mar 2024
Corrected Phase 2	19 Mar 2024

Source Files

Filename	SHA256
contracts/ReusedCoinerec.sol	c7782b71868a6fea6533624318418d341cc16335fd736cfc3aac3c55dde9b480

Findings Breakdown



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

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner, as described in detail in section PTRP. As a result, the contract may operate as a honeypot.

Recommendation

The team is strongly encouraged to adhere to the recommendations outlined in the respective sections. By doing so, the contract can eliminate any potential of operating as a honeypot.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L521,536
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function removeAllFee() private { ... }
function restoreAllFee() private { ... }
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L380
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.

```
uniswapV2Pair
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L626
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tokenAmount) private {
    /* Generate the Uniswap pair path of token -> weth */
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router), tokenAmount);

    /* Make the swap */
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L642,673
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
transferToAddressETH(marketingWallet, convertedBalance);
function transferToAddressETH(address payable recipient, uint256 amount)
private {
    sendValue(recipient,amount);
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L832
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 setMinSell(uint256 amount) external onlyOwner {
    require(amount < _tTotal, 'Should be less than total supply');
    numTokensSellToAddToLiquidity = amount;
    emit SetMinSellEvent(amount);
}
```

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 exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L371
Status	Unresolved

Description

The contract uses events that 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 TransferToMarketingWalletFailed(string message);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L580,582
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
/* Add liquidity */
swapAndLiquify(liquidityBalance);
/* Send Marketing Fees */
swapTokens(marketingBalance);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L342,349
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 _liquidityDenominator = 1000
uint256 private _marketingDenominator = 1000
```

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	contracts/ReusedCoinerec.sol#L283,332,333,334,335,337,338,343,344,451,842,848,855
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 constant _tTotal = 100 * 10 ** 12 * 10 ** 18
string constant _name = 'ReusedCoin'
string constant _symbol = 'REUSED'
uint8 constant _decimals = 18
uint256 public _liquidityFee = 20
uint256 public _liquidityBuyFee = 10
uint256 public _marketingFee = 20
uint256 public _marketingBuyFee = 20
address _owner
bool _enabled
address _to
address _token
uint256 _amount
```

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.

L09 - Dead Code Elimination

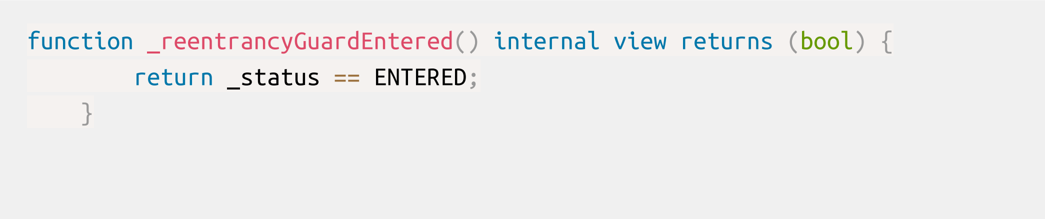
Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L249
Status	Unresolved

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 _reentrancyGuardEntered() internal view returns (bool) {
        return _status == ENTERED;
    }
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/ReusedCoinerec.sol#L451,551
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.

```
address _owner
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	geUnlockTime	Public		-
	lock	Public	✓	onlyOwner
	unlock	Public	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

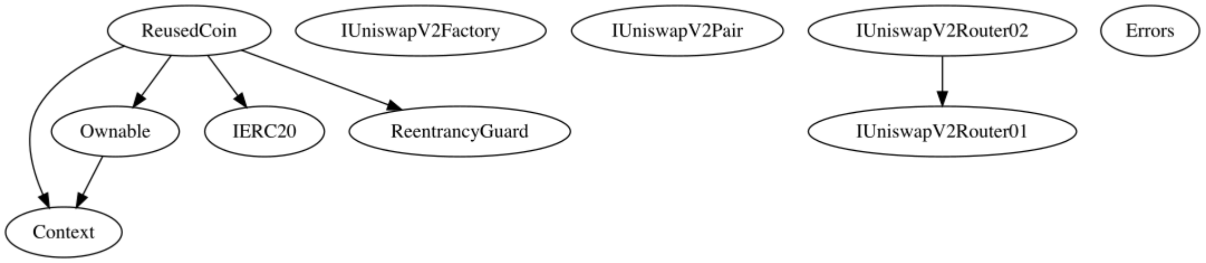
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
	_reentrancyGuardEntered	Internal		
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	factory	External		-
	token0	External		-
	token1	External		-

IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Errors	Library			
ReusedCoin	Implementation	Context, IERC20, Ownable, ReentrancyGuard		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	nonReentrant
	allowance	Public		-
	approve	Public	✓	-

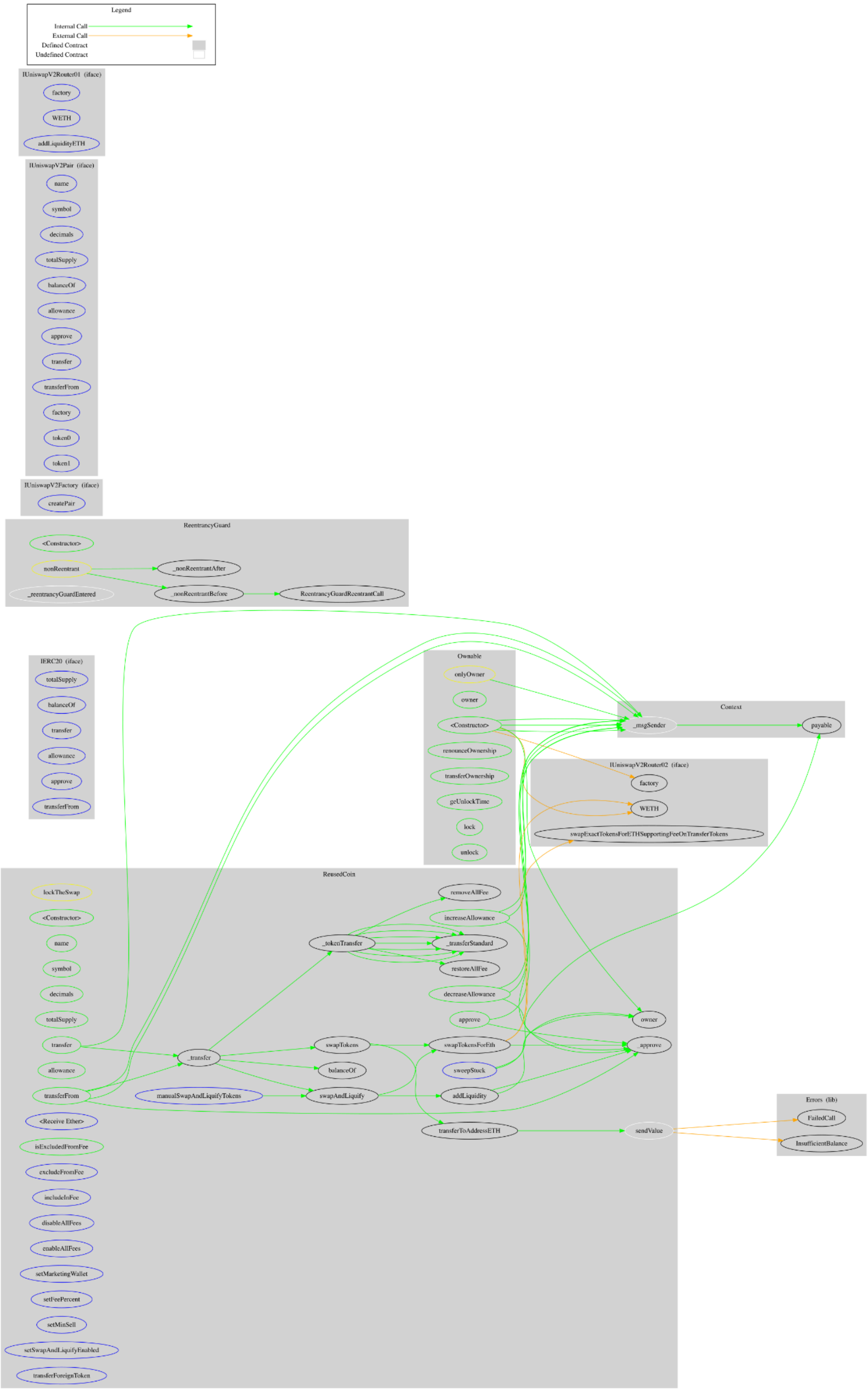
	transferFrom	Public	✓	nonReentrant
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
		External	Payable	-
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	swapTokens	Private	✓	lockTheSwap
	sendValue	Internal	✓	
	transferToAddressETH	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	excludeFromFee	External	✓	onlyOwner
	includeInFee	External	✓	onlyOwner
	disableAllFees	External	✓	onlyOwner
	enableAllFees	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setFeePercent	External	✓	onlyOwner

	setMinSell	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	transferForeignToken	External	✓	onlyOwner
	sweepStuck	External	✓	onlyOwner
	manualSwapAndLiquifyTokens	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01 (iface)
factory
WETH
addLiquidityETH
IUniswapV2Pair (iface)
name
symbol
decimals
totalSupply
balanceOf
allowance
approve
transfer
transferFrom
factory
token0
token1
IUniswapV2Factory (iface)
createPair
ReentrancyGuard
<Constructor>
nonReentrant
_nonReentrantAfter
_reentrancyGuardEntered
_nonReentrantBefore
ReentrancyGuardReentrantCall
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
transferOwnership
geUnlockTime
lock
unlock
Context
_msgSender
payable
IUniswapV2Router02 (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
ReusedCoin
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
transfer
allowance
transferFrom
<Receive Ether>
isExcludedFromFee
excludeFromFee
includeInFee
disableAllFees
enableAllFees
setMarketingWallet
setFeePercent
setMinSell
setSwapAndLiquifyEnabled
transferForeignToken
_tokenTransfer
_transfer
manualSwapAndLiquifyTokens
swapTokens
balanceOf
swapAndLiquify
removeAllFee
increaseAllowance
_transferStandard
restoreAllFee
decreaseAllowance
approve
swapTokensForEth
sweepStuck
addLiquidity
transferToAddressETH
owner
_approve
sendValue
Errors (lib)
FailedCall
InsufficientBalance

Summary

ReusedCoin 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 stopping transactions. 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 25% buy and 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