Cyberscope

Audit Report

KhabyWorld

January 2023

Type	ERC20
Network	ETH
Address	0x3Eb63F33eB01cb39a0ce711844FaA8b581cCbEa0
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Analysis 5
ST - Stops Transactions 6
Description 6
Recommendation 6
ELFM - Exceeds Fees Limit 7
Description 7
Recommendation 7
OCTD - Transfers Contract's Tokens 8
Description 8
Recommendation 8
Diagnostics 9
CO - Code Optimization 10
Description 10
Recommendation 10
PVC - Price Volatility Concern 11
Description 11
Recommendation 11
ZD - Zero Division 12
Description 12
Recommendation 12
L02 - State Variables could be Declared Constant 13
Description 13
Recommendation 13
L04 - Conformance to Solidity Naming Conventions 14
Description 14
Recommendation 15
L07 - Missing Events Arithmetic 16

Description 16
Recommendation 16
L09 - Dead Code Elimination 17
Description 17
Recommendation 18
L13 - Divide before Multiply Operation 19
Description 19
Recommendation 19
L15 - Local Scope Variable Shadowing 20
Description 20
Recommendation 20
L16 - Validate Variable Setters 21
Description 21
Recommendation 21
Functions Analysis 22
Inheritance Graph 27
Flow Graph 28
Summary 29
Disclaimer 30
About Cyberscope 31

Review

Contract Name	KWorld
Compiler Version	v0.8.12+commit.f00d7308
Optimization	1 runs
Explorer	https://etherscan.io/address/0x3eb63f33eb01cb39a0ce711844faa8b581ccbea0
Address	0x3eb63f33eb01cb39a0ce711844faa8b581ccbea0
Network	ETH
Symbol	$KWorld
Decimals	18
Total Supply	1,000,000,000

Audit Updates

Initial Audit	05 Jan 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	75e3c97011e75627ffb36f4a2799a4e887e1a3e27ed427490e82d7b6f51cc5c9
@openzeppelin/contracts/token/ERC20/ERC20.sol	3cd9bf87ad804088f574a5266771f038a2c44b53d85f355aadb35645e497d1c2
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
contracts/Token.sol	a9d8974aad8c1fb635df43cd7879ea578002e7e57814c7de7549bfc8ca4481e9

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	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	Passed

ST - Stops Transactions

Criticality	Critical
Location	contracts/Token.sol#L437,445
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the

- `maxTxAmount` to zero.
- `maxWalletBalance` to a very low value (minimum is 1).

As a result, the contract may operate as a honeypot.

```
require(amount <= maxTxAmount, "Transfer amount exceeds the Max Transaction Amount.");
...
require(recipientBalance + amount <= maxWalletBalance, "New balance would exceed the maxWalletBalance");
```

Recommendation

The contract could embody a check for not allowing setting the `maxTxAmount` and the `maxWalletBalance` less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/Token.sol#L579
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 `setTaxFeePercent` function with a high percentage value.

```
function setTaxes(uint256 _devSellFee, uint256 _rewardSellFee, uint256
_marketingSellFee, uint256 _devBuyFee, uint256 _marketingBuyFee, uint256 _LPBuyFee)
external onlyOwner {
  devBuyFee = _devBuyFee;
  devSellFee = _devSellFee;
  rewardSellFee = _rewardSellFee;
  marketingSellFee = _marketingSellFee;
  marketingBuyFee = _marketingBuyFee;
  LPBuyFee = _LPBuyFee;
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/Token.sol#L575
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `recoverERC20Token` function.

```
function recoverERC20Token(address tokenAddress, uint256 tokens) external onlyOwner
returns (bool success){
  return ERC20(tokenAddress).transfer(msg.sender, tokens);
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	ZD	Zero Division	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/Token.sol#L422
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. Since the variables `tradingActive` and `tradingActiveBlock` never change, the `if-statement` can be removed as the condition will always resolve to false after the contract's deployment.

```
if(!tradingActive || tradingActiveBlock + 2 >= block.number){
  require(_isExcludedFromFees[from] || _isExcludedFromFees[to], "Trading is always
active after deployment");
}
```

Recommendation

The team is advised to take into consideration these segments and remove 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	contracts/Token.sol#L342
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 returns
(bool){
  swapTokensAtAmount = newAmount * (10**18);
  return true;
}
```

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.

ZD - Zero Division

Criticality	Medium
Location	contracts/Token.sol#L464,472
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 `totalSellFees` and `totalBuyFees` variables are the calculated sum of the fees. If all the fee values are set to zero, the result will be a division by zero.

```
uint256 totalSellFees = marketingSellFee + devSellFee + rewardSellFee;
tokensForMarketing += fees * marketingSellFee / totalSellFees;
...
uint256 totalBuyFees = marketingBuyFee + devBuyFee + LPBuyFee;
tokensForMarketing += fees * marketingBuyFee / totalBuyFees;
```

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 executing of the corresponding statements.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/Token.sol#L235,237,242
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 tradingActiveBlock = 0
bool public tradingActive = true
uint256 public feeDivisor = 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	contracts/Token.sol#L29,30,47,85,226,253,269,270,579,588,592
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
address public DevWallet
uint256 public LPBuyFee
event marketingWalletUpdated(address indexed newWallet, address indexed oldWallet);
event rewardWalletUpdated(address indexed newWallet, address indexed oldWallet);
uint256 _devBuyFee
uint256 _rewardSellFee
uint256 _LPBuyFee
uint256 _marketingSellFee
uint256 _devSellFee
uint256 _marketingBuyFee

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/Token.sol#L343,580,589,593
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
swapTokensAtAmount = newAmount * (10**18)
devBuyFee = _devBuyFee
maxWalletBalance = _maxWalletAmount
maxTxAmount = _maxTxAmount
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/Token.sol#L489
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 swapEthForNativeToken(uint256 ethAmount) private {
        if(ethAmount > 0){
            address[] memory path = new address[](2);
            path[0] = uniswapV3Router.WETH();
            path[1] = address(this);

            uniswapV3Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
ethAmount}(
                0,
                path,
                address(marketingWallet),
                block.timestamp
            );
        }
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/Token.sol#L464,465,466,471,472,473,474
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
fees = amount * totalBuyFees / feeDivisor
tokensForLP += fees * LPBuyFee / totalBuyFees
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/Token.sol#L281
Status	Unresolved

Description

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

```
uint256 totalSupply = 1_000_000_000 * (10**18)
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/Token.sol#L300,304,306,308
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.

```
marketingWallet = _marketingWallet
DevWallet = _DevWallet
autoLiquidityReceiver = _newOwner
rewardWallet = _rewardWallet
```

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**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	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	✓	

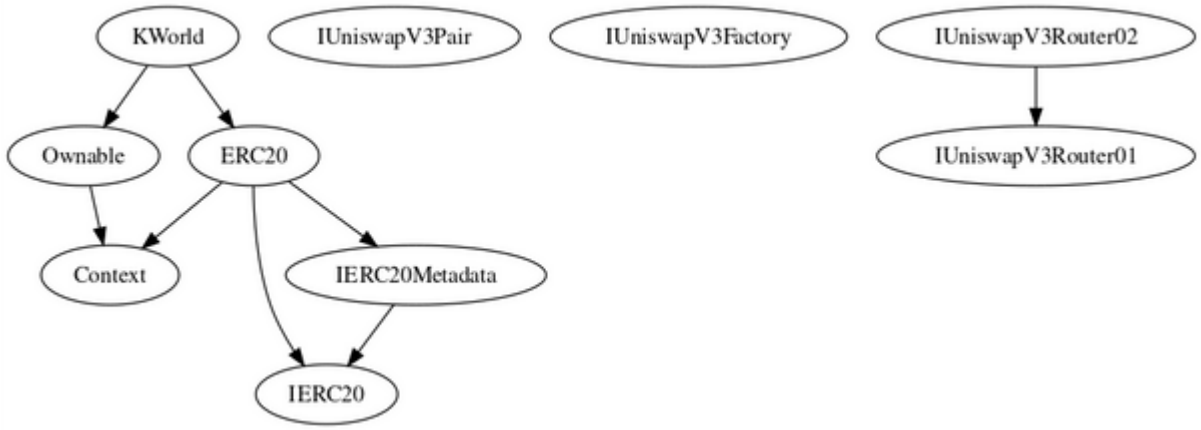
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV3Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-

	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV3Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-

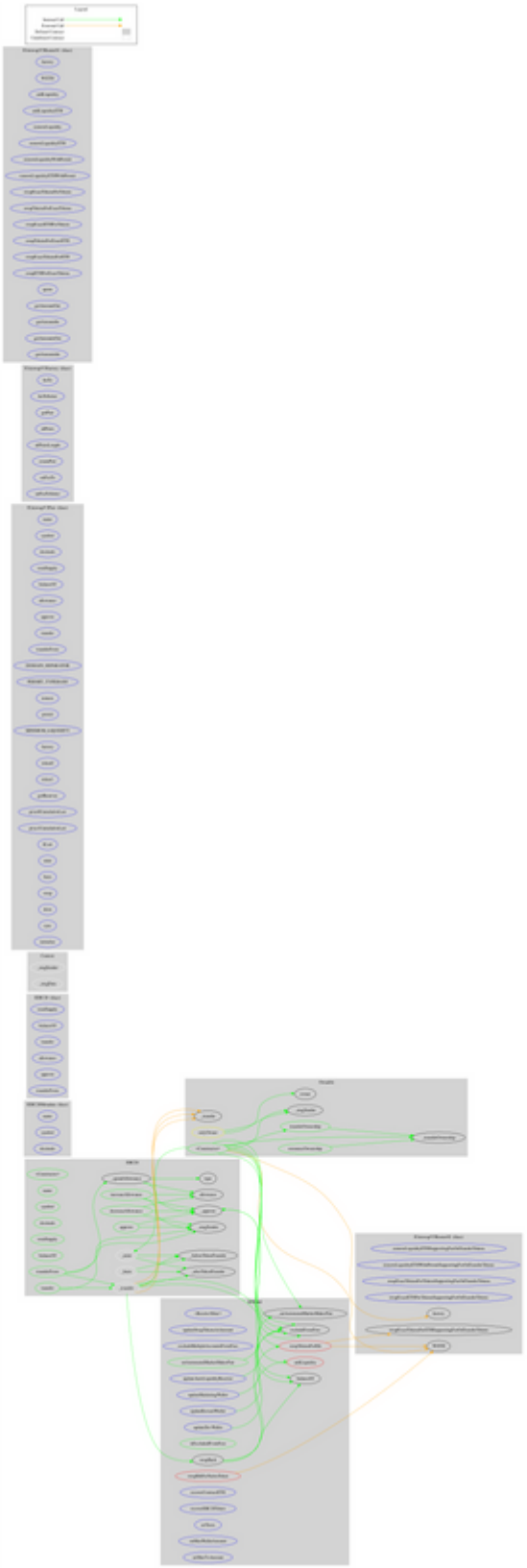
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV3Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV3Router02	Interface	IUniswapV3Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-

	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
KWorld	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateSwapTokensAtAmount	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateAutoLiquidityReceiver	External	✓	onlyOwner
	updateMarketingWallet	External	✓	onlyOwner
	updateRewardWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapEthForNativeToken	Private	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	recoverContractETH	External	✓	onlyOwner
	recoverERC20Token	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setMaxWalletAmount	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

There are some functions that can be abused by the owner like stop transactions, drain the contract's tokens and manipulate the fees. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. 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 100% buy/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