Cyberscope

Audit Report

YouCantWin

December 2023

Network BSC

Address 0x1df540d1e0f514210ac3efd921913c5054af4559

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	RIFT	Redundant Intermediate Funds Transfer	Unresolved
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	DDP	Decimal Division Precision	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	RRS	Redundant Require Statement	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L05	Unused State Variable	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L15	Local Scope Variable Shadowing	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
- Audit Updates 5
- Source Files 5

Findings Breakdown 6
- CR - Code Repetition 7
  - Description 7
  - Recommendation 7
- RIFT - Redundant Intermediate Funds Transfer 8
  - Description 8
  - Recommendation 8
- RSW - Redundant Storage Writes 10
  - Description 10
  - Recommendation 10
- RRS - Redundant Require Statement 11
  - Description 11
  - Recommendation 11
- RSML - Redundant SafeMath Library 12
  - Description 12
  - Recommendation 12
- L04 - Conformance to Solidity Naming Conventions 13
  - Description 13
  - Recommendation 14
- L05 - Unused State Variable 15
  - Description 15
  - Recommendation 15
- L07 - Missing Events Arithmetic 16
  - Description 16
  - Recommendation 16
- L09 - Dead Code Elimination 17
  - Description 17
  - Recommendation 17
- L15 - Local Scope Variable Shadowing 19
  - Description 19
  - Recommendation 19
- L16 - Validate Variable Setters 20
  - Description 20

Recommendation 20
L20 - Succeeded Transfer Check 21
Description 21
Recommendation 21
Functions Analysis 22
Inheritance Graph 30
Flow Graph 31
Summary 32
Disclaimer 33
About Cyberscope 34

Review

Contract Name	YouCantWin
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0x1df540d1e0f514210ac3efd921913c5054af4559
Address	0x1df540d1e0f514210ac3efd921913c5054af4559
Network	BSC
Symbol	Win
Decimals	18
Total Supply	100,000,000

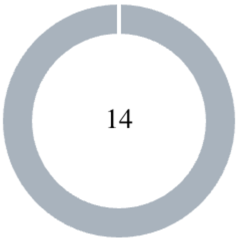
Audit Updates

Initial Audit	07 Dec 2023

Source Files

Filename	SHA256
YouCantWin.sol	aa3584be43fab66d1baa530697cc8c11e018b89fc73a42ea121c6187d90b2cf7

Findings Breakdown



- Critical 0
- Medium 0
- Minor / Informative 14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

CR - Code Repetition

Criticality	Minor / Informative
Location	YouCantWin.sol#L1131
Status	Unresolved

Description

The contract contains repetitive code segments. The code checks whether the transaction involves an automated market maker (AMM) pair and, if so, calculates fees based on the specified fee structures for liquidity and marketing. This results in redundant code blocks for the sell and buy paths, leading to increased maintenance complexity and potential risk of inconsistencies between the two sections.

```
if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
    fees = amount.mul(sellTotalFees).div(100);
    tokensForLiquidity += fees * sellLiquidityFee / sellTotalFees;
    tokensForMarketing += fees * sellMarketingFee / sellTotalFees;
}
// on buy
else if(automatedMarketMakerPairs[from] && buyTotalFees > 0) {
    fees = amount.mul(buyTotalFees).div(100);
    tokensForLiquidity += fees * buyLiquidityFee / buyTotalFees;
    tokensForMarketing += fees * buyMarketingFee / buyTotalFees;
}
```

Recommendation

To improve code maintainability and reduce redundancy, consider refactoring the code to consolidate the common fee calculation logic for both sell and buy scenarios. By creating a shared function or merging the common elements, the code becomes more concise and less prone to errors or inconsistencies. This refactoring can enhance readability and simplify future updates or modifications to the fee calculation logic.

This refactoring approach centralizes the fee calculation logic, reducing code repetition and making the contract more maintainable in the long run.

RIFT - Redundant Intermediate Funds Transfer

Criticality	Minor / Informative
Location	YouCantWin.sol#L1188
Status	Unresolved

Description

The swapBack function, funds are initially sent to the marketing wallet, and subsequently, the leftover funds are transferred again in a later step. This intermediate transfer to the marketing wallet appears redundant and could be optimized to improve the contract's efficiency.

```
function swapBack() private {
    // ... (previous code)

    // Intermediate transfer to marketing wallet
    (bool success,) = address(marketingWallet).call{value:
ethForMarketing}("");

    if (liquidityTokens > 0 && ethForLiquidity > 0) {
        // ... (liquidity-related operations)
        emit SwapAndLiquify(amountToSwapForETH, ethForLiquidity,
tokensForLiquidity);
    }

    // Redundant transfer of leftover ETH to marketing wallet
    (success,) = address(marketingWallet).call{value:
address(this).balance}("");
}
```

Recommendation

To enhance the clarity and efficiency of the smart contract, it is recommended to eliminate the intermediate transfer of funds to the marketing wallet. Instead, consolidate the fund transfer operations at the end of the function. This can be achieved by accumulating the leftover ETH and sending it to the marketing wallet in a single step.

This modification simplifies the logic and reduces the number of transactions, potentially optimizing gas consumption while maintaining the intended functionality of the smart contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	YouCantWin.sol#L1004
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function excludeFromMaxTransaction(address updAds, bool
isEx) public onlyOwner {
        _isExcludedMaxTransactionAmount[updAds] = isEx;
    }

    function excludeFromFees(address account, bool excluded)
public onlyOwner {
        _isExcludedFromFees[account] = excluded;
        emit ExcludeFromFees(account, excluded);
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	YouCantWin.sol#L467
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
	uint256 c = a + b;
	require(c >= a, "SafeMath: addition overflow");
	return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	YouCantWin.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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	YouCantWin.sol#L31,32,49,731,899,911,990,1013,1020,1224
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);
mapping (address => bool) public
_isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
bool _enable
uint256 _liquidityFee
uint256 _marketingFee
address _address
```

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	YouCantWin.sol#L653
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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	YouCantWin.sol#L984,996,1001,1014,1021
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
maxWallet = newNum * (10**18)
maxTransactionAmount = newNum * (10**18)
buyMarketingFee = _marketingFee
sellMarketingFee = _marketingFee
```

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	YouCantWin.sol#L398,699,705,712
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 _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount,
"ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

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	YouCantWin.sol#L943
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 = 100000000 * 1e18
```

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	YouCantWin.sol#L1046
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 = newMarketingWallet
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	YouCantWin.sol#L1228
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(_address).transfer(owner(),
IERC20(_address).balanceOf(address(this)))
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV2Pair	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	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-

	setFeeToSetter	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
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	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		

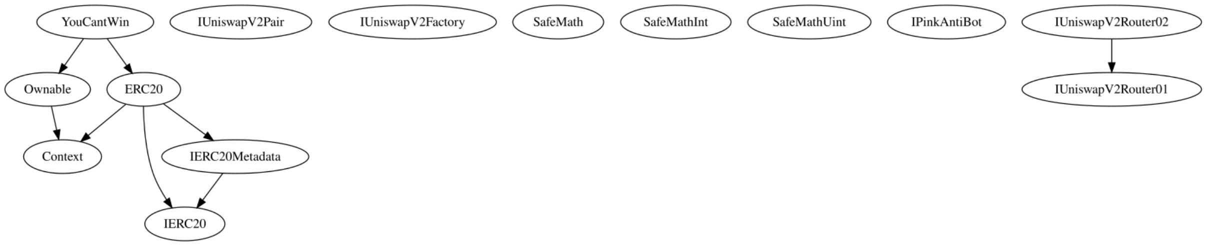
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IPinkAntiBot	Interface			
	setTokenOwner	External	✓	-
	onPreTransferCheck	External	✓	-
IUniswapV2Router01	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		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

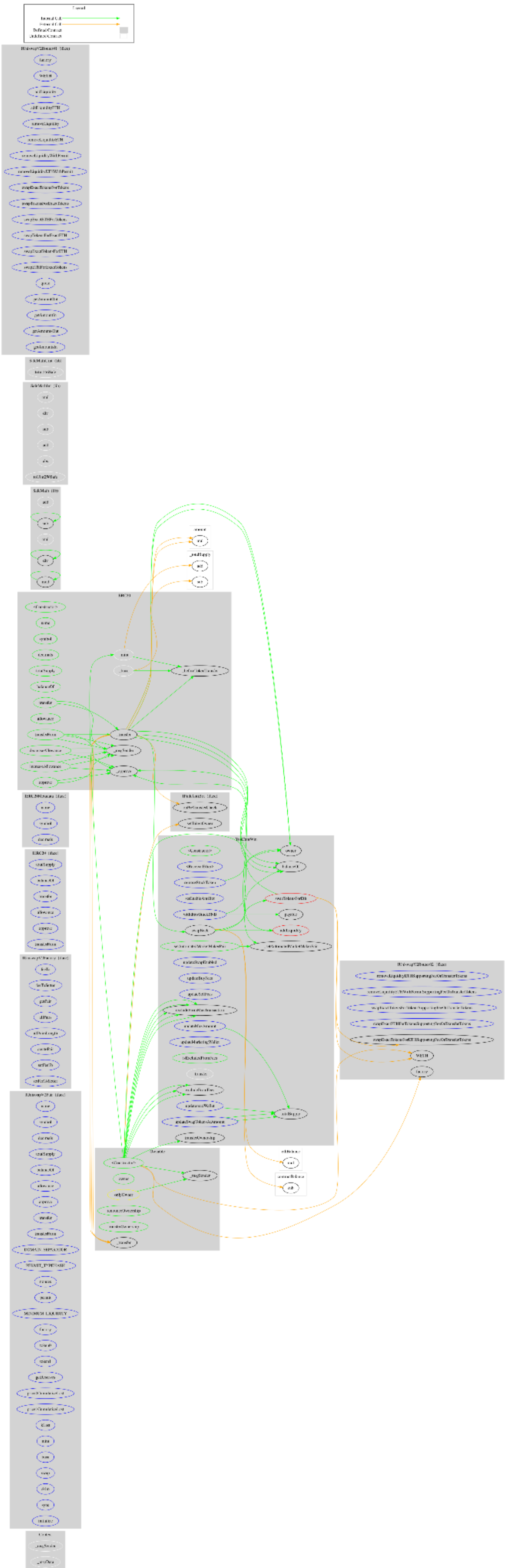
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
YouCantWin	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateSwapTokensAtAmount	External	✓	onlyOwner
	setEnableAntiBot	External	✓	onlyOwner
	updatemaxWallet	External	✓	onlyOwner
	updateMaxAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	

	withdrawStuckBNB	External	✓	onlyOwner
	removeStuckToken	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

YouCantWin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. YouCantWin is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 15% 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