Cyberscope

Audit Report

MiniBNBTiger

April 2023

Network	BSC
Address	0x51213757F952D333Fb4240356d67acDcEf38645A
Audited by	© cyberscope

Table of Contents

Review

Contract Name	MiniBNBTiger
Compiler Version	v0.7.4+commit.3f05b770
Optimization	200 runs
Explorer	https://bscscan.com/address/0x51213757f952d333fb4240356d67acdcef38645a
Address	0x51213757f952d333fb4240356d67acdcef38645a
Network	BSC
Symbol	MiniBNBTiger
Decimals	9
Total Supply	10,000,000,000,000,000,000,000,000

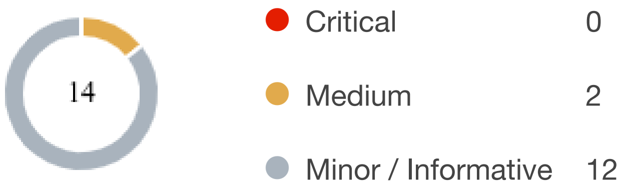
Audit Updates

Initial Audit	24 Apr 2023

Source Files

Filename	SHA256
MiniBNBTiger.sol	0c270f11e919c6fca96feb9e5f98c8ee3dd24847ecd149936e44b2968647dbfd

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	2	0	0	0
● Minor / Informative	12	0	0	0

Analysis

🔴 Critical 🟡 Medium ⚪ Minor / Informative 🔵 Pass

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

ST - Stops Transactions

Criticality	Medium
Location	MiniBNBTiger.sol#L510,542
Status	Unresolved

Description

The contract authorized users have the authority to stop the transactions for all users excluding the authorized users. The authorized users may take advantage of it by setting the `_maxTxAmount` to zero.

```
checkTxLimit(sender, amount);
...
function checkTxLimit(address sender, uint256 amount) internal view {
    require(amount <= _maxTxAmount || isTxLimitExempt[sender], "TX Limit
Exceeded");
}
```

Recommendation

The contract could embody a check for not allowing setting the _maxTxAmount 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. 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.

ELFM - Exceeds Fees Limit

Criticality	Medium
Status	Unresolved

Description

The contract authorized users have the authority to increase over the allowed limit of 25%. The authorized users may take advantage of it by calling the `setFees` function with a high percentage value.

```
function setFees(uint256 _liquidityFee, uint256 _reflectionFee, uint256
_marketingFee, uint256 _devfee, uint256 _burnFee, uint256 _feeDenominator) external
authorized {
    liquidityFee = _liquidityFee;
    reflectionFee = _reflectionFee;
    marketingFee = _marketingFee;
    devfee = _devfee;
    burnFee = _burnFee;
    totalFee =
_liquidityFee.add(_reflectionFee).add(_marketingFee).add(_devfee).add(_burnFee);
    feeDenominator = _feeDenominator;
    require(totalFee < feeDenominator/2, "Fees cannot be more than 50%");
}
```

Recommendation

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

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	PVC	Price Volatility Concern	Unresolved
●	ZD	Zero Division	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L704
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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 setSwapBackSettings(bool _enabled, uint256 _amount) external
authorized {
    swapEnabled = _enabled;
    swapThreshold = _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 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	Minor / Informative
Location	MiniBNBTiger.sol#L627
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 `totalBNBFee` can be zero if `totalFee` is equal to zero or equal to `dynamicLiquidityFee` and the rest of the fees are zero. As a result, the transaction will revert.

```
uint256 amountBNBLiquidity =
amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L627,628,629,630
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 amountBNBLiquidity =
amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
uint256 amountBNBReflection =
amountBNB.mul(reflectionFee).div(totalBNBFee);
uint256 amountBNBMarketing = amountBNB.mul(marketingFee).div(totalBNBFee);
uint256 amountBNBDev = amountBNB.mul(devfee).div(totalBNBFee);
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L414,415,418
Status	Unresolved

Description

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

```
router
pair
distributor
```

Recommendation

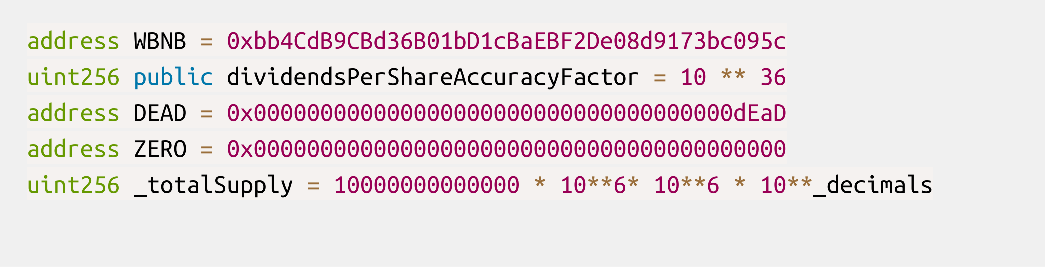
By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L192,205,349,350,351,357
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.



```
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
uint256 _totalSupply = 10000000000000 * 10**6* 10**6 * 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	MiniBNBTiger.sol#L127,183,191,192,230,349,350,351,353,354,355,357,359,360,362,363,472,475,581,586,591,599,663,667,686,697,704,709,714,764
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);
address _token
IBEP20 RWRD = IBEP20(0x5a04565ee1c90c84061aD357AE9E2f1c32D57dc6)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 _minDistribution
uint256 _minPeriod
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
string constant _name = "MiniBNBTiger"
string constant _symbol = "MiniBNBTiger"
uint8 constant _decimals = 9
uint256 _totalSupply = 10000000000000 * 10**6* 10**6 * 10**_decimals
uint256 public _maxTxAmount = _totalSupply.mul(2).div(100)
uint256 public _maxWalletToken = _totalSupply.mul(2).div(100)

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, and 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	MiniBNBTiger.sol#L231,476,480,587,687,706,710
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.

```
minPeriod = _minPeriod
_maxTxAmount = (_totalSupply * maxTXPercentage_base1000 ) / 1000
_maxTxAmount = amount
sellMultiplier = Multiplier
liquidityFee = _liquidityFee
swapThreshold = _amount
targetLiquidity = _target
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L553,555
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 a loss of prediction.

```
uint256 feeAmount =
amount.mul(totalFee).mul(multiplier).div(feeDenominator * 100)
uint256 burnTokens = feeAmount.mul(burnFee).div(totalFee)
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L668
Status	Unresolved

Description

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

```
uint256 i
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L113,698,699,700,701
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.

```
owner = adr
autoLiquidityReceiver = _autoLiquidityReceiver
marketingFeeReceiver = _marketingFeeReceiver
devfeeReceiver = _devfeeReceiver
burnFeeReceiver = _burnFeeReceiver
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L22
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.7.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	MiniBNBTiger.sol#L308
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.

```
RWRD.transfer(shareholder, amount)
```

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**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

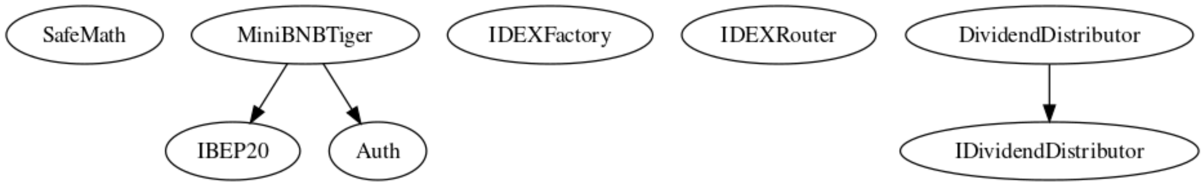
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	authorize	Public	✓	onlyOwner
	unauthorize	Public	✓	onlyOwner
	isOwner	Public		-
	isAuthorized	Public		-
	transferOwnership	Public	✓	onlyOwner
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-

IDividendDistributor	Interface			
	setDistributionCriteria	External	✓	-
	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-
DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
MiniBNBTiger	Implementation	IBEP20, Auth		
		Public	✓	Auth
		External	Payable	-

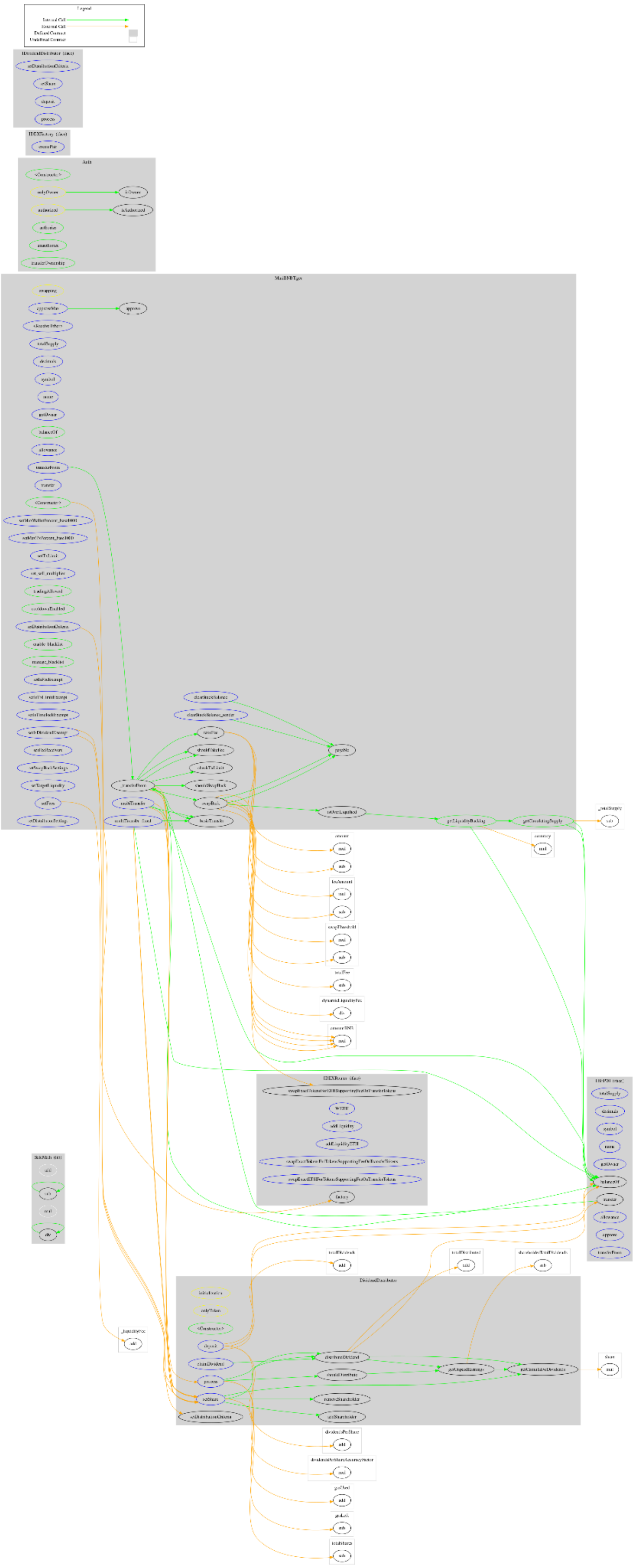
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	setMaxWalletPercent_base1000	External	✓	onlyOwner
	setMaxTxPercent_base1000	External	✓	onlyOwner
	setTxLimit	External	✓	authorized
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	checkTxLimit	Internal		
	shouldTakeFee	Internal		
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	clearStuckBalance	External	✓	authorized
	clearStuckBalance_sender	External	✓	authorized
	set_sell_multiplier	External	✓	onlyOwner

	tradingAllowed	Public	✓	onlyOwner
	cooldownEnabled	Public	✓	onlyOwner
	swapBack	Internal	✓	swapping
	setIsDividendExempt	External	✓	authorized
	enable_blacklist	Public	✓	onlyOwner
	manage_blacklist	Public	✓	onlyOwner
	setIsFeeExempt	External	✓	authorized
	setIsTxLimitExempt	External	✓	authorized
	setIsTimelockExempt	External	✓	authorized
	setFees	External	✓	authorized
	setFeeReceivers	External	✓	authorized
	setSwapBackSettings	External	✓	authorized
	setTargetLiquidity	External	✓	authorized
	setDistributionCriteria	External	✓	authorized
	setDistributorSettings	External	✓	authorized
	getCirculatingSupply	Public		-
	getLiquidityBacking	Public		-
	isOverLiquified	Public		-
	multiTransfer	External	✓	onlyOwner
	multiTransfer_fixed	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

MiniBNBTiger contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The contract limits the sales to one sale per 30 seconds. There are some functions that can be abused by the owner like stopping transactions and manipulating the fees. 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 limit of max 50% fees.

The contract has renounced the ownership but the authorized addresses cannot be renounced. Hence, the address 0xabd598c3692d12e53b9cfafe07ff817f95d92fe4 has access to all of the authorized methods.

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