Cyberscope

Audit Report

Red Dragon

January 2024

Network	BSC
Address	0x7fA0bc115B5608a4aDfBb695a4127287fb96EE9a
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ZD	Zero Division	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MN	Misleading Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RES	Redundant Event Statement	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	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
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	Token
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x7fa0bc115b5608a4adfbb695a4127287fb96ee9a
Address	0x7fa0bc115b5608a4adfbb695a4127287fb96ee9a
Network	BSC
Symbol	Red
Decimals	18
Total Supply	21,000,000,000,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	28 Jan 2024

Source Files

Filename	SHA256
Token.sol	949057023d09ca1bb001db6b362f4f39303dba79db0f9adde86bbcbcbbf33ec2

Findings Breakdown

Critical	4
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Token.sol#L599,618
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` or `_walletMax` to zero. As a result, the contract may operate as a honeypot.

```
if(!isTxLimitExempt[sender] && !isTxLimitExempt[recipient]) {
    require(amount <= _maxTxAmount, "Transfer amount exceeds
the maxTxAmount.");
}

if(checkWalletLimit && !isWalletLimitExempt[recipient])
    require(balanceOf(recipient).add(finalAmount) <=
_walletMax);
```

Additionally, the contract owner has the authority to stop the sales, as described in detail in sections ZD and PTRP. As a result, the contract may operate as a honeypot in this way as well.

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` and the `_walletMax` less than a reasonable amount. A suggested implementation could check that the minimum 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

OTUT - Transfers User's Tokens

Criticality	Critical
Location	Token.sol#L589
Status	Unresolved

Description

The contract owner has the authority to transfer liquidity to the owner's address. Specifically, function `_transfer` includes a conditional clause that checks if the sender is `uniswapPair` and current block is less than the number of the variable `first` and another variable `kill`. The `kill` variable can be modified by the contract owner through the `setKing` function. When this condition is met, funds are redirected to `receiverAddress`, which is the address of the contract owner.

```
if(sender == uniswapPair && block.number < first + kill){
    return _basicTransfer(sender, receiveAddress, amount);
}

function setKing(uint256 newValue) public onlyOwner {
    kill = newValue;
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Token.sol#L456,474
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 `setSellDestFee` and `setSelTaxes` functions with a high percentage value.

```
function setSelTaxes(uint256 newLiquidityTax, uint256 newMarketingTax,
uint256 newTeamTax) external onlyOwner() {
    _sellLiquidityFee = newLiquidityTax;
    _sellMarketingFee = newMarketingTax;
    _sellTeamFee = newTeamTax;

    _totalTaxIfSelling =
_sellLiquidityFee.add(_sellMarketingFee).add(_sellTeamFee).add(_sellDestr
oyFee);
    }

function setSellDestFee(uint256 newSellDestroyFee) public onlyOwner {
    _sellDestroyFee = newSellDestroyFee;
    _totalTaxIfSelling =
_sellLiquidityFee.add(_sellMarketingFee).add(_sellTeamFee).add(_sellDestr
oyFee);
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ZD - Zero Division

Criticality	Critical
Location	Token.sol#L636
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. Specifically, the contract owner has the authority to set `_totalDistributionShares` to zero, by calling `setDistributionSettings` with all arguments set to zero. As a result the contract may operate as a honeypot.

```
uint256 tokensForLP =
tAmount.mul(_liquidityShare).div(_totalDistributionShares).div(
2);


function setDistributionSettings(uint256 newLiquidityShare,
uint256 newMarketingShare, uint256 newTeamShare) external
onlyOwner() {
    _liquidityShare = newLiquidityShare;
    _marketingShare = newMarketingShare;
    _teamShare = newTeamShare;

    _totalDistributionShares =
_liquidityShare.add(_marketingShare).add(_teamShare);
    }
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Token.sol#L345,347,349
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.

```
_decimals
receiveAddress
_totalSupply
```

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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	Token.sol#L614
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(balanceOf(recipient).add(finalAmount) <= _walletMax)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MN - Misleading Naming

Criticality	Minor / Informative
Location	Token.sol#L295,524
Status	Unresolved

Description

Methods and variables can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names and variables that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve.

```
uint256 public kill = 0;

function setKing(uint256 newValue) public onlyOwner {
    kill = newValue;
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Token.sol#L435,439,443,447,451,456,461,469...
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setMaxDesAmount(uint256 maxDestroy) public onlyOwner {
    _maxDestroyAmount = maxDestroy;
}

function setBuyTaxes(uint256 newLiquidityTax, uint256
newMarketingTax, uint256 newTeamTax) external onlyOwner() {
    _buyLiquidityFee = newLiquidityTax;
    _buyMarketingFee = newMarketingTax;
    _buyTeamFee = newTeamTax;

    _totalTaxIfBuying =
_buyLiquidityFee.add(_buyMarketingFee).add(_buyTeamFee).add(_buyDestr
oyFee);
}

...
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	Token.sol#L547
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function changeRouterVersion(address newRouterAddress) public
onlyOwner returns(address newPairAddress) {

    IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(newRouterAddress);

    newPairAddress =
IUniswapV2Factory(_uniswapV2Router.factory()).getPair(address(t
his), _uniswapV2Router.WETH());

    if(newPairAddress == address(0)) //Create If Doesnt exist
    {
        newPairAddress =
IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this),
_uniswapV2Router.WETH());
    }

    uniswapPair = newPairAddress; //Set new pair address
    uniswapV2Router = _uniswapV2Router; //Set new router
address

    isWalletLimitExempt[address(uniswapPair)] = true;
    isMarketPair[address(uniswapPair)] = true;
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` and `teamWalletAddress` as part of the transfer flow. Those addresses 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 and the contract may operate as a honeypot.

```
transferToAddressETH(marketingWalletAddress,
amountBNBMarketing);

transferToAddressETH(teamWalletAddress, amountBNBTeam);

function transferToAddressETH(address payable recipient,
uint256 amount) private {
    recipient.transfer(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.

RES - Redundant Event Statement

Criticality	Minor / Informative
Location	Token.sol#L307,313
Status	Unresolved

Description

Event `SwapAndLiquify` and `SwapETHForTokens` are declared but never used within the contract's functions. Events in Solidity are crucial for emitting information from the contract to the outside world, particularly to off-chain applications and user interfaces. They are essential for tracking contract activity and providing transparency in transactions. The declaration of these events without corresponding emit statements in functions where deposits and withdrawals occur represents a redundancy in the contract code. This omission could lead to a lack of vital transactional data being available to users or off-chain services.

```
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiqudity
);

event SwapETHForTokens(
    uint256 amountIn,
    address[] path
);
```

Recommendation

It is recommended that the contract be updated to include emit statements for these events, if it's required. If the contract's design intentionally omits the use of this event, it would be advisable to remove its declaration to streamline the contract and avoid confusion. Proper implementation and use of events are vital for maintaining the integrity and transparency of smart contract operations.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Token.sol#L43
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
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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Token.sol#L435,439,443,447,451,456,461,469...
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 setIsTxLimitExempt(address holder, bool exempt)
external onlyOwner {
    isTxLimitExempt[holder] = exempt;
}

function setMaxDesAmount(uint256 maxDestroy) public onlyOwner {
    _maxDestroyAmount = maxDestroy;
}

...
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Token.sol#L254
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 public deadAddress =
0x000000000000000000000000000000000000dEaD
```

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	Token.sol#L147,189,264,265,266,267,269,270,271,272,274,275,276,277,279,280,282,283,285,286,515
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.

```
address public _owner
function WETH() external pure returns (address);
uint256 public _buyLiquidityFee = 2
uint256 public _buyMarketingFee = 3
uint256 public _buyTeamFee = 4
uint256 public _buyDestroyFee = 0
uint256 public _sellLiquidityFee = 2
uint256 public _sellMarketingFee = 3
uint256 public _sellTeamFee = 4
uint256 public _sellDestroyFee = 0
uint256 public _liquidityShare = 2
uint256 public _marketingShare = 3
uint256 public _teamShare = 4
uint256 public _totalDistributionShares = 9

...
```

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	Token.sol#L444,448,453,462,467,475,479,487,499,503,521
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.

```
_maxDestroyAmount = maxDestroy
_buyDestroyFee = newBuyDestroyFee
_sellDestroyFee = newSellDestroyFee
_totalTaxIfBuying =
_buyLiquidityFee.add(_buyMarketingFee).add(_buyTeamFee).add(_buyDestroyF
ee)
airdropNumbs = newValue
_totalTaxIfSelling =
_sellLiquidityFee.add(_sellMarketingFee).add(_sellTeamFee).add(_sellDest
royFee)
_liquidityShare = newLiquidityShare
_maxTxAmount = maxTxAmount
_walletMax  = newLimit
_minimumTokensBeforeSwap = newLimit
kill = newValue
```

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	Token.sol#L90,101,109,113,117,121,126
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 isContract(address account) internal view returns
(bool) {
        // According to EIP-1052, 0x0 is the value returned for
not-yet created accounts
        // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470;
        // solhint-disable-next-line no-inline-assembly
        assembly {codehash := extcodehash(account)}
        return (codehash != accountHash && codehash != 0x0);
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Token.sol#L346,354,355,376,508,512
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 = owner
marketingWalletAddress = payable(marketingAddress)
teamWalletAddress = payable(teamAddress)
payable(service).transfer(msg.value)
marketingWalletAddress = payable(newAddress)
teamWalletAddress = payable(newAddress)
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Token.sol#L97,135
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {codehash := extcodehash(account)}

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Token.sol#L2
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 ^0.8.4;
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

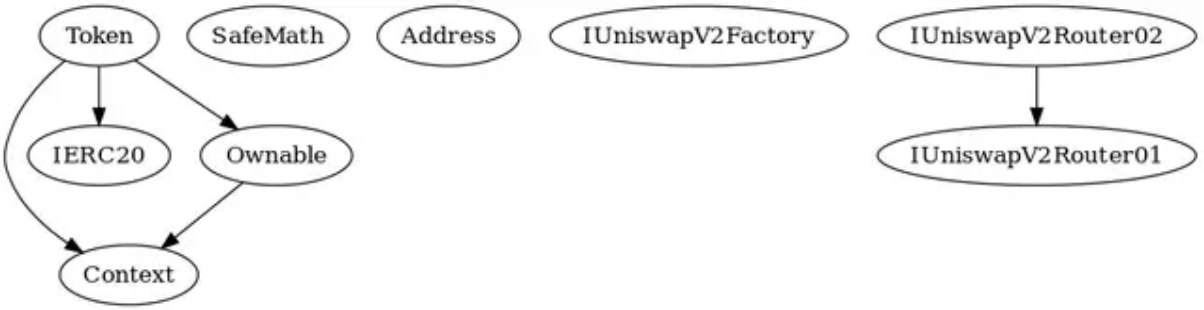
	mod	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
	owner	Public		-
	waiveOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	getTime	Public		-
IUniswapV2Factory	Interface			
	getPair	External		-
	createPair	External	✓	-
IUniswapV2Router01	Interface			

	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Token	Implementation	Context, IERC20, Ownable		
		Public	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	minimumTokensBeforeSwapAmount	Public		-
	approve	Public	✓	-

	_approve	Private	✓	
	setMarketPairStatus	Public	✓	onlyOwner
	setIsTxLimitExempt	External	✓	onlyOwner
	setIsExcludedFromFee	Public	✓	onlyOwner
	setMaxDesAmount	Public	✓	onlyOwner
	setBuyDestFee	Public	✓	onlyOwner
	setSellDestFee	Public	✓	onlyOwner
	setBuyTaxes	External	✓	onlyOwner
	setAirdropNumbs	Public	✓	onlyOwner
	setSelTaxes	External	✓	onlyOwner
	setDistributionSettings	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	enableDisableWalletLimit	External	✓	onlyOwner
	setIsWalletLimitExempt	External	✓	onlyOwner
	setWalletLimit	External	✓	onlyOwner
	setNumTokensBeforeSwap	External	✓	onlyOwner
	setMarketingWalletAddress	External	✓	onlyOwner
	setTeamWalletAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setKing	Public	✓	onlyOwner
	setSwapAndLiquifyByLimitOnly	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	getCirculatingSupply	Public		-

	transferToAddressETH	Private	✓	
	changeRouterVersion	Public	✓	onlyOwner
		External	Payable	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Private	✓	
	_basicTransfer	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	takeFee	Internal	✓	
	destroyFee	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
IUniswapV2Factory
getPair
createPair
Address
sendValue
functionCall
functionCallWithValue
isContract
SafeMath
add
sub
mul
div
mod
IERC20
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Token
lockTheSwap
transferFrom
name
symbol
decimals
totalSupply
allowance
minimumTokensBeforeSwapAmount
setMarketPairStatus
setIsTxLimitExempt
setIsExcludedFromFee
setMaxTxAmount
transfer
setDistributionSettings
setBuyDevsFee
setApiAvoyNumbs
setBuyTaxes
setSellDevsFee
setMaxTxAmount
enableDisableWalletLimit
setIsWalletLimitExempt
setWalletLimit
setNumTokensBeforeSwap
setSellTaxes
setSwapAndLiquifyEnabled
setDog
setSwapAndLiquifyByLimitOnly
excludeMultipleAccountsFromFees
changeRouterVersion
setMarketingWalletAddress
setTeamWalletAddress
getCirculatingSupply
addLiquidity
approve
transferToAddressETH
increaseAllowance
decreaseAllowance
swapTokensForEth
destroyFee
takeFee
swapAndLiquify
basicTransfer
balanceOf
transfer
totalFees
feeAmount
Context
_msgSender
_msgData
payable
Ownable
owner
onlyOwner
waiveOwnership
transferOwnership
getTime
liquidityShare
buyLiquidityFee
sellLiquidityFee
IUniswapV2Router02
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory

Summary

Red Dragon contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions, transfer the user's tokens and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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