Cyberscope

Audit Report

BitRocket23

April 2023

SHA256 22f74e604fe6514c21c6a7e40c11cd78d46d8eeb87d88e4e63dbbf25c33991d9

Audited by © cyberscope

Table of Contents

Review

Contract Name	BitRocket23
Testing Deploy	https://testnet.bscscan.com/address/0xa5f8dbad7dd8ccfddfa7bebf4e4ccf8459c167c9
Symbol	BRCKVNUS23
Decimals	9
Total Supply	10,000,000,000

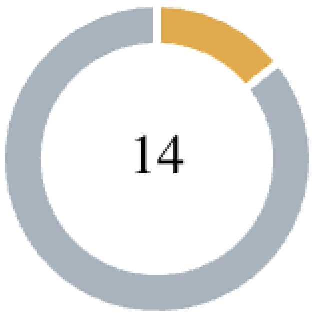
Audit Updates

Initial Audit	29 Apr 2023

Source Files

Filename	SHA256
contracts/testingDeploy/V23_-_Contract_2023.sol	22f74e604fe6514c21c6a7e40c11cd78d46d8eeb87d88e4e63dbbf25c33991d9

Findings Breakdown



●	Critical	0
●	Medium	2
●	Minor / Informative	12

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	contracts/testingDeploy/V23_-_Contract_2023.sol#L1049
Status	Unresolved

Description

Initially, the contract does not allow the non-excluded addresses to transfer tokens. The restriction can be resumed once the contract owner enable them.

```
if(!_tradingEnabled && from != owner() ) {
    if(!_isExcludedFromFee[from])
    {
        revert("Trading not yet enabled!");
    }
}
```

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

ELFM - Exceeds Fees Limit

Criticality	Medium
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 not calling the `completeFairLaunch()` method. As a result the transfers will be taxed by 99%.

```
if(!_fairLaunchCompleted)
{
    _reflectionFee = 99;
}
```

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	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
●	L20	Succeeded Transfer Check	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/testingDeploy/V23_-_Contract_2023.sol#L673
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.

The `if` branch and the `isAdmin/isManager` variable declaration is reduntant since it is not used by the contract.

```
modifier contractAdmin() {
        bool isAdmin = false;
        if(isContractAdmin(_msgSender())  || isOwner())
        {
            isAdmin = true;
        }
        require(isContractAdmin(_msgSender())  || isOwner(),
"Admin: caller is not a contract Administrator");
        _;
    }

    modifier contractManager() {
        bool isManager = false;
        if(isContractManager(_msgSender())  || isOwner())
        {
            isManager = true;
        }
        require(isContractManager(_msgSender())  || isOwner(),
"Manager: caller is not a contract Manager");
        _;
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite 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/testingDeploy/V23_-_Contract_2023.sol#L1234
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 setnumTokensSwapToETHForMarketing(uint256 tokenAmount)
external contractAdmin() {
    _numTokensSwapToETHForMarketing = tokenAmount;
}
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/testingDeploy/V23_-_Contract_2023.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 on 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 the 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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/testingDeploy/V23_-_Contract_2023.sol#L403,404,742,746,747,748
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 private _previousOwner
uint256 private _lockTime
uint256 private _tTotal = 10 * 10**9 * 10**9
string private _name = "BRCKVNUS23"
string private _symbol = "BRCKVNUS23"
uint8 private _decimals = 9
```

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/testingDeploy/V23_-_Contract_2023.sol#L492,493,510,532,750,752,754,757,763,764,765,767,966,976,986,1229,1249
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);
uint8 public _reflectionFee = 1
uint8 public _marketingFee = 1
uint8 public _marketingSellFee = 3
address public MarketingSellFeeWallet =
0x5ae4C9540A2eb9Ba42031eE4873cB449c9097dBa
bool public _marketingConverttoETH = true
bool public _tradingEnabled = false
bool public _fairLaunchCompleted = false
uint256 public _maxTxAmount = 10 * 10**6 * 10**9
uint256 _amount
bool _enabled

...
```

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	contracts/testingDeploy/V23_-_Contract_2023.sol#L403,404
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.

```
address private _previousOwner
uint256 private _lockTime
```

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	contracts/testingDeploy/V23_-_Contract_2023.sol#L1213,1235
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.

```
_maxTxAmount = maxAmountInTokensWithDecimals
_numTokensSwapToETHForMarketing = tokenAmount
```

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/testingDeploy/V23_-_Contract_2023.sol#L272,299,325,335,350,360,365
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
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // 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	contracts/testingDeploy/V23_-_Contract_2023.sol#L1204,1208
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.

```
MarketingSellFeeWallet = newWallet
marketingFeeWallet = newWallet
```

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	contracts/testingDeploy/V23_-_Contract_2023.sol#L279,378
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	contracts/testingDeploy/V23_-_Contract_2023.sol#L7
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 ^0.8.14;
```

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	contracts/testingDeploy/V23_-_Contract_2023.sol#L1250
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(_tokenAddr).transfer(_to, _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**
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		

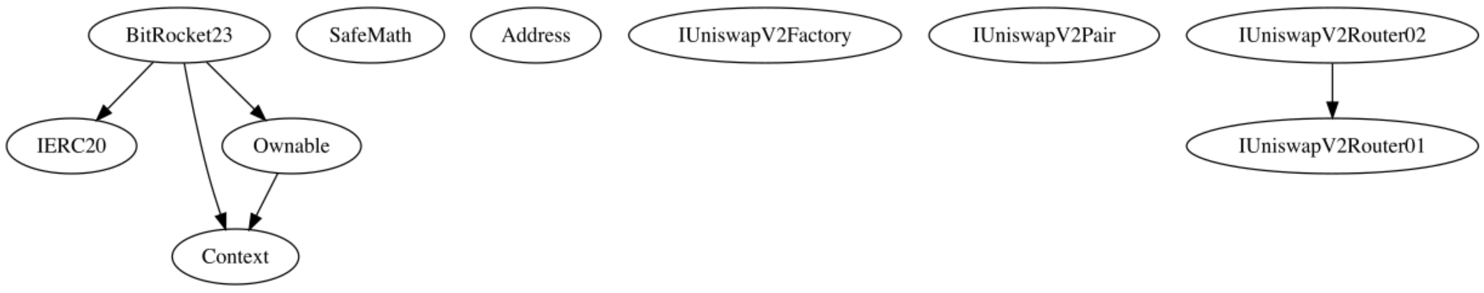
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
BitRocket23	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-

	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalReflectionFees	Public		-
	numTokensSwapToETHForMarketing	Public		-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	contractManager
		External	Payable	-
	_distributeFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeMarketingSellFee	Private	✓	

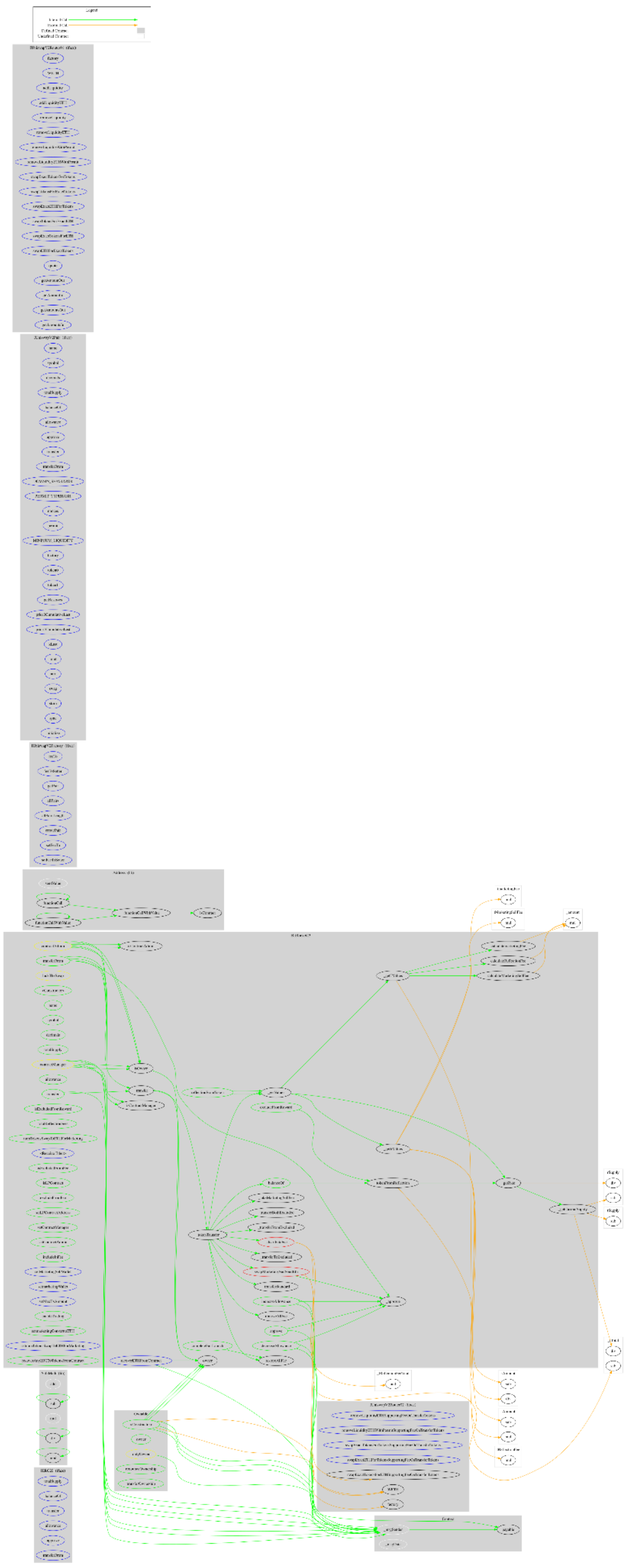
	calculateReflectionFee	Private		
	calculatemarketingFee	Private		
	calculateMarketingSellFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	isLPContract	Public		-
	isContractAdmin	Public		-
	isContractManager	Public		-
	isOwner	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapMarketingAndSendEth	Private	✓	lockTheSwap
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
	_transferBothExcluded	Private	✓	
	excludeFromFee	Public	✓	contractAdmin
	setLPContractAddress	Public	✓	contractAdmin
	setContractManager	Public	✓	contractManager
	setContractAdmin	Public	✓	contractManager
	includeInFee	Public	✓	contractAdmin

	setMarketingSellWallet	External	✓	contractAdmin
	setmarketingWallet	External	✓	contractAdmin
	setMaxTxAmount	External	✓	contractAdmin
	enableTrading	Public	✓	onlyOwner
	completeFairLaunch	Public	✓	onlyOwner
	setmarketingConverttoETH	Public	✓	contractAdmin
	setnumTokensSwapToETHForMarketing	External	✓	contractAdmin
	recoverETHFromContract	External	✓	contractManager
	recoverAnyERC20TokensFromContract	Public	✓	contractManager

Inheritance Graph



Flow Graph



Summary

BitRocket23 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 and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. The fees are fixed to 5%.

Disclaimer

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