



Cyberscope

# Audit Report

## **Blokramp**

February 2023

Network     BSC

Address     0x16985295bdd0337c18eb11e0fc39696d4200a5bc

Audited by   © cyberscope

# Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Review</b>	<b>2</b>
<b>Audit Updates</b>	<b>2</b>
<b>Source Files</b>	<b>2</b>
<b>Analysis</b>	<b>3</b>
<b>Diagnostics</b>	<b>4</b>
<b>L02 - State Variables could be Declared Constant</b>	<b>5</b>
Description	5
Recommendation	5
<b>L04 - Conformance to Solidity Naming Conventions</b>	<b>6</b>
Description	6
Recommendation	6
<b>L05 - Unused State Variable</b>	<b>8</b>
Description	8
Recommendation	8
<b>L12 - Using Variables before Declaration</b>	<b>9</b>
Description	9
Recommendation	9
<b>L14 - Uninitialized Variables in Local Scope</b>	<b>10</b>
Description	10
Recommendation	10
<b>L16 - Validate Variable Setters</b>	<b>11</b>
Description	11
Recommendation	11
<b>Functions Analysis</b>	<b>12</b>
<b>Inheritance Graph</b>	<b>15</b>
<b>Flow Graph</b>	<b>16</b>
<b>Summary</b>	<b>17</b>
<b>Disclaimer</b>	<b>18</b>
<b>About Cyberscope</b>	<b>19</b>

## Review

<b>Contract Name</b>	Blokrapm
<b>Compiler Version</b>	v0.8.18+commit.87f61d96
<b>Optimization</b>	500 runs
<b>Explorer</b>	<a href="https://bscscan.com/address/0x16985295bdd0337c18eb11e0fc39696d4200a5bc">https://bscscan.com/address/0x16985295bdd0337c18eb11e0fc39696d4200a5bc</a>
<b>Address</b>	0x16985295bdd0337c18eb11e0fc39696d4200a5bc
<b>Network</b>	BSC
<b>Symbol</b>	BRMP
<b>Decimals</b>	18
<b>Total Supply</b>	100,000,000

## Audit Updates

<b>Initial Audit</b>	11 Feb 2023
<b>Corrected Phase 2</b>	12 Feb 2023

## Source Files

<b>Filename</b>	SHA256
<b>Blokrapm.sol</b>	e1907a9e6b3ee6b7764e4845c63024179 cc3ec2be6b56d092c6b0f900cefa3b5

# Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

# Diagnostics

● Critical   ● Medium   ● Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved

## L02 - State Variables could be Declared Constant

<b>Criticality</b>	Minor / Informative
<b>Location</b>	Blokram.sol#L102,115
<b>Status</b>	Unresolved

### Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private timeSinceLastPair = 0  
bool public taxesAreLocked
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	Blokrapm.sol#L33,109,110,111,112,113,121,273
<b>Status</b>	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);
uint256 constant private startingSupply = 100_000_000
string constant private _name = "Blokrapm"
string constant private _symbol = "BRMP"
uint8 constant private _decimals = 18
uint256 constant private _tTotal = startingSupply * 10**_decimals
bool public _hasLiqBeenAdded = false
bool _antiSnipe
bool _antiBlock
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	Blokramp.sol#L102,106
<b>Status</b>	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.

```
uint256 private timeSinceLastPair = 0  
mapping (address => bool) private _isExcludedFromLimits
```

### 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.

## L12 - Using Variables before Declaration

<b>Criticality</b>	Minor / Informative
<b>Location</b>	Blokram.sol#L368
<b>Status</b>	Unresolved

### Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
bool check
```

### Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

## L14 - Uninitialized Variables in Local Scope

<b>Criticality</b>	Minor / Informative
<b>Location</b>	Blokrapm.sol#L367,368
<b>Status</b>	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.

```
bool checked  
bool check
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	Blokram.sol#L197
<b>Status</b>	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.

```
operator = newOperator
```

### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

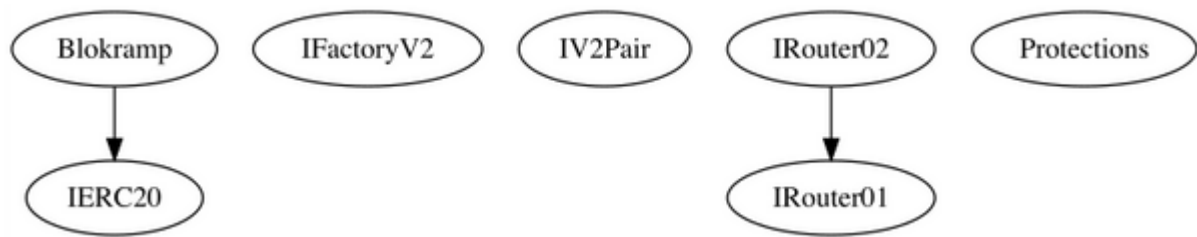
# Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>IERC20</b>	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
<b>IFactoryV2</b>	Interface			
	getPair	External		-
	createPair	External	✓	-
<b>IV2Pair</b>	Interface			
	factory	External		-
	getReserves	External		-
	sync	External	✓	-
<b>IRouter01</b>	Interface			
	factory	External		-
	WETH	External		-

	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
<b>IRouter02</b>	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
<b>Protections</b>	Interface			
	checkUser	External	✓	-
	setLaunch	External	✓	-
	setLpPair	External	✓	-
	setProtections	External	✓	-
	removeSniper	External	✓	-
<b>Blokram</b>	Implementation	IERC20		
		Public	Payable	-
		External	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	setOperator	Public	✓	-
	renounceOriginalDeployer	External	✓	-
	totalSupply	External		-
	decimals	External		-

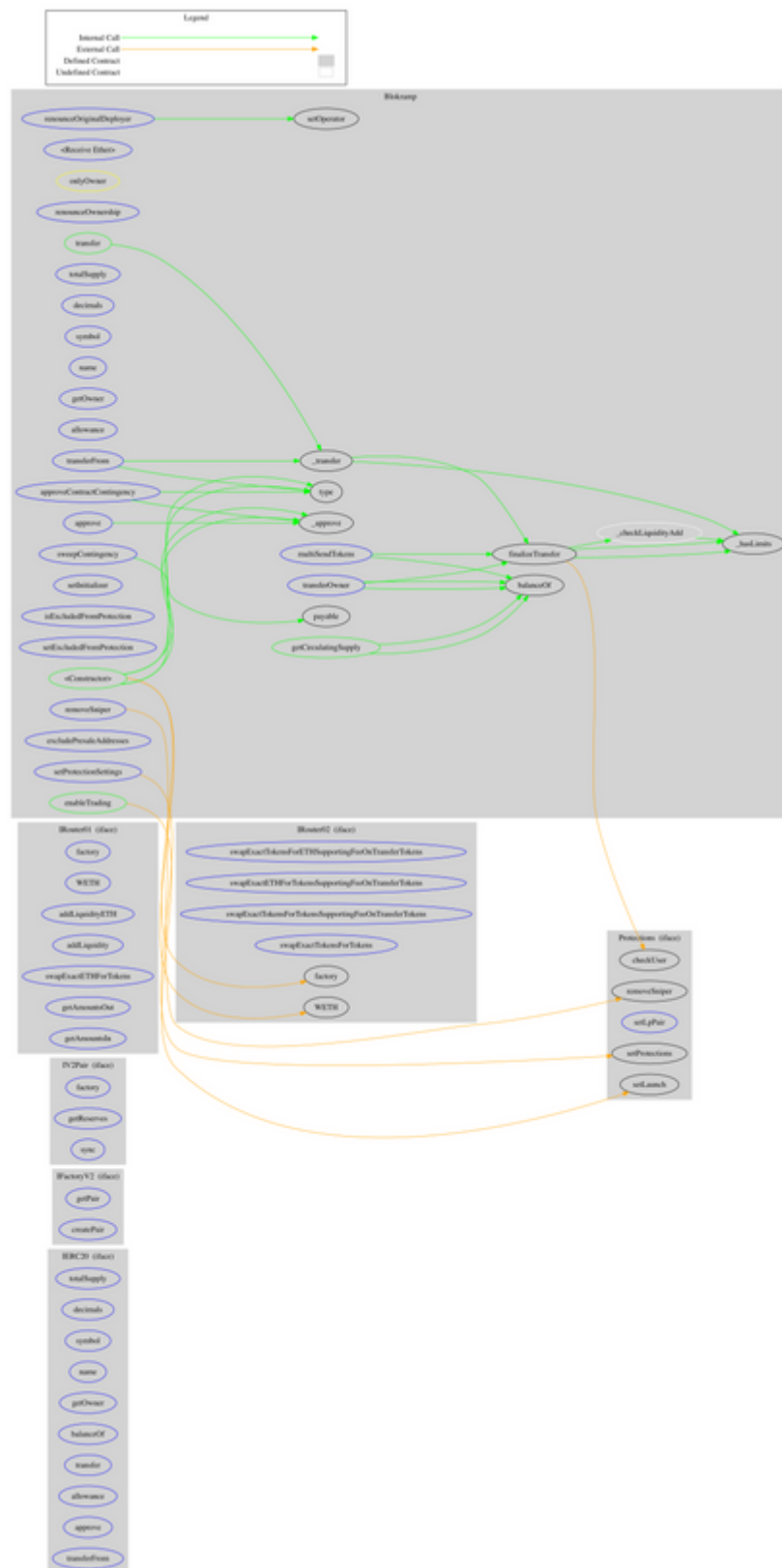
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	External	✓	onlyOwner
	transferFrom	External	✓	-
	setInitializer	External	✓	onlyOwner
	isExcludedFromProtection	External		-
	setExcludedFromProtection	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner
	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_transfer	Internal	✓	
	_checkLiquidityAdd	Internal	✓	
	enableTrading	Public	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	multiSendTokens	External	✓	onlyOwner
	finalizeTransfer	Internal	✓	

## Inheritance Graph





# Flow Graph



## Summary

Blokram is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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.

## 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>