Cyberscope

Audit Report

ObeseFans Calories

June 2023

Network	BSC
Address	0x69830556232e56a6a33672ad6b8e5937d22ce90c
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Calories
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x69830556232e56a6a33672ad6b8e5937d22ce90c
Address	0x69830556232e56a6a33672ad6b8e5937d22ce90c
Network	BSC
Symbol	$CLRS
Decimals	18
Total Supply	10.000.000.000

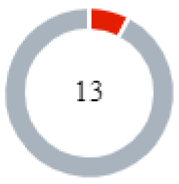
Audit Updates

Initial Audit	11 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v1/audit.pdf
Corrected Phase 2	12 May 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v2/audit.pdf
Corrected Phase 3	13 Jun 2023

Source Files

Filename	SHA256
Calories.sol	7855b0ee4e3b6ad2e1d7383baa43a287bae20c15703b1150211fb170f0080da9

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	12	0	0	0

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

BT - Burns Tokens

Criticality	Critical
Location	contracts/CLRS.sol#L1351
Status	Unresolved

Description

The contract automatically burns up to 10% of the liquidity pair token balance every 10 minutes at most. If a large amount of liquidity is removed from the pool through burning, it can cause a decrease in the liquidity of the pool, which can, in turn, result in increased volatility and price fluctuations of the tokens in the pair.

```
function _autoBurnLiquidityPairTokens() internal returns
(bool){

  lastLpBurnTime = block.timestamp;

  // get balance of liquidity pair
  uint256 liquidityPairBalance = this.balanceOf(uniswapV2Pair);

  // calculate amount to burn
  uint256 amountToBurn = liquidityPairBalance *
percentForLPBurn / 10000;

  // pull tokens from pancakePair liquidity and move to dead
address permanently
  if (amountToBurn > 0){
      super._transfer(uniswapV2Pair, address(0xdead),
amountToBurn);
  }

  //sync price since this is not in a swap transaction!
  IUniswapV2Pair pair = IUniswapV2Pair(uniswapV2Pair);
  pair.sync();
  emit AutoNukeLP();
  return true;
}
```

Recommendation

It is recommended to review and adjust the parameters of the auto-liquidity burn mechanism to ensure that it operates optimally. Specifically, the period of time and percentage burned should be reasonable and appropriate for the specific use case. This will help to prevent any potential issues and ensure that the mechanism functions as intended.

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
●	OCTD	Transfers Contract's Tokens	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Unresolved
●	RE	Redundant Event	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1130
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueAnyBEP20Tokens` function.

```
function rescueAnyBEP20Tokens(address _tokenAddr, address _to,
uint _amount) external
  onlyOwner
  AddressNotZero(_tokenAddr)
  AddressNotZero(_to) {
  uint balance = IERC20(_tokenAddr).balanceOf(address(this));
  _amount = _amount > balance ? balance : _amount;
  if(_tokenAddr == address(this)){
      balance -= _amount;
      // reset the token balances for new balance
      uint totalFees = sellTotalFees;
      tokensForCharity = balance * sellCharityFee / totalFees;
      tokensForDev = balance * sellDevFee / totalFees;
      tokensForMarketing = balance * sellMarketingFee /
totalFees;
      tokensForLiquidity = balance * sellLiquidityFee /
totalFees;
  }

  IERC20(_tokenAddr).transfer(_to, _amount);
}
```

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. 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.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	Calories.sol#L1125
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `rescueBNB` methods.

```
function rescueBNB(uint256 weiAmount) external onlyOwner{
  require(address(this).balance >= weiAmount, "insufficient BNB
balance");
  payable(msg.sender).transfer(weiAmount);
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. 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.

RE - Redundant Event

Criticality	Minor / Informative
Location	NFTPresale.sol#L955
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 `BoughtEarly` event is not utilized in the contract implementation. Hecne, it is redundant.

```
event BoughtEarly(address indexed sniper);
```

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. It is recommended to remove redundant events.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Calories.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Calories.sol#L50,51,68,741,938,940,942,956,1070,1079,1130,1343
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);
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event devWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event charityWalletUpdated(address indexed newWallet, address
indexed oldWallet);

modifier AddressNotZero(address value) {
        if (value == address(0)) revert AddressIsZero();
        _;
    }
uint256 _liquidityFee
uint256 _devFee
uint256 _buyCharityFee

...
```

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	Calories.sol#L672
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Calories.sol#L1047,1053,1058,1071,1080,1140,1346
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
swapTokensAtAmount = newAmount
maxTransactionAmount = newNum * (10**18)
maxWallet = newNum * (10**18)
buyMarketingFee = _marketingFee
sellMarketingFee = _marketingFee
tokensForCharity = balance * sellCharityFee / totalFees
lpBurnFrequency = _frequencyInSeconds
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	Calories.sol#L1345
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_percent <= 1000 && _percent >= 0, "Must set auto LP
burn percent between 0% and 10%")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Calories.sol#L417,718,724,731
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

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

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

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

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Calories.sol#L1237,1238,1239,1240,1244,1245,1246,1247,1248
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 loss of prediction.

```
tokensForLiquidity += fees * sellLiquidityFee / sellTotalFees
fees = amount * buyTotalFees / 100
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Calories.sol#L982
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 totalSupply = 1 * 1e10 * 1e18
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Calories.sol#L1146
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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-

	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-

	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-

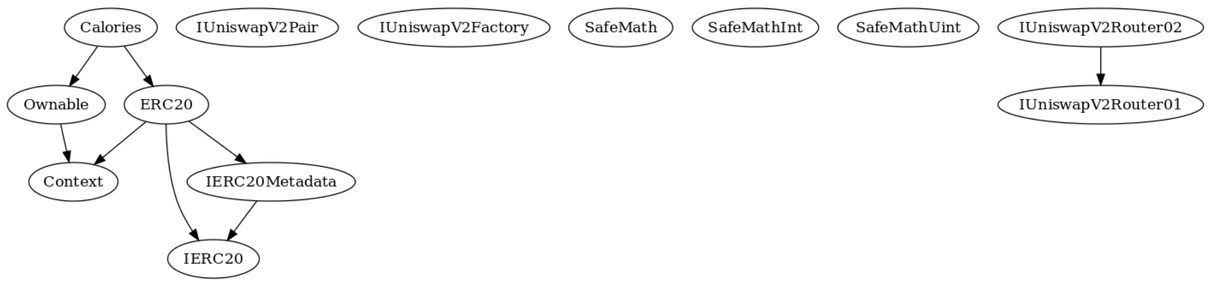
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-

	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Calories	Implementation	ERC20, Ownable		
		Public	✓	ERC20

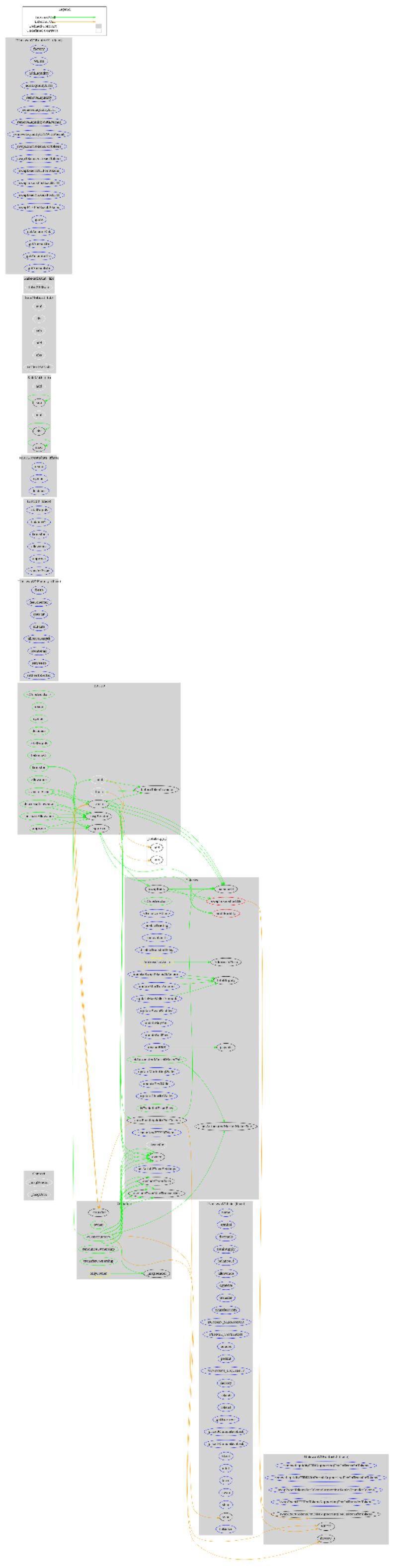
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner AddressNotZero
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner AddressNotZero
	setAutomatedMarketMakerPair	Public	✓	onlyOwner AddressNotZero
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner AddressNotZero
	updateDevWallet	External	✓	onlyOwner AddressNotZero
	updateCharityWallet	External	✓	onlyOwner AddressNotZero
	isExcludedFromFees	Public		-
	rescueBNB	External	✓	onlyOwner
	rescueAnyBEP20Tokens	External	✓	onlyOwner AddressNotZer

				o AddressNotZer o
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	setAutoLPBurnSettings	External	✓	onlyOwner
	_autoBurnLiquidityPairTokens	Internal	✓	

Inheritance Graph



Flow Graph



Summary

ObeseFans Calories 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 burning tokens from any address. if the contract owner abuses the burning functionality, the users could lose their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 20% fee.

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