Cyberscope

Audit Report

ObeseFans Calories

April 2023

Network	BSC Testnet
Address	0xe94005A06fa7713d24ac992dDf01bD8D7e08DfCA
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Findings Breakdown **4**
Analysis **5**
OCTD - Transfers Contract's Tokens 6
Description 6
Recommendation 6
ULTW - Transfers Liquidity to Team Wallet 7
Description 7
Recommendation 7
BT - Burns Tokens 8
Description 8
Recommendation 9
Diagnostics **10**
AFI - Accumulated Fees Inconsistency 11
Description 11
Recommendation 11
RSML - Redundant SafeMath Library 12
Description 12
Recommendation 12
L02 - State Variables could be Declared Constant 13
Description 13
Recommendation 13
L04 - Conformance to Solidity Naming Conventions 14
Description 14
Recommendation 15
L05 - Unused State Variable 16
Description 16
Recommendation 16
L07 - Missing Events Arithmetic 17
Description 17
Recommendation 17
L08 - Tautology or Contradiction 18
Description 18
Recommendation 18
L09 - Dead Code Elimination 19
Description 19

Recommendation 19
L13 - Divide before Multiply Operation 21
Description 21
Recommendation 21
L15 - Local Scope Variable Shadowing 22
Description 22
Recommendation 22
L16 - Validate Variable Setters 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Explorer	https://testnet.bscscan.com/address/0xe94005a06fa7713d24ac992ddf01bd8d7e08dfca

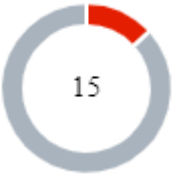
Audit Updates

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

Source Files

Filename	SHA256
Calories.sol	c9bcbb3edf12928f5098e98e3d5fe1da3b1a3eb3e311f0d0f432e7e4d03464ca

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Analysis

Critical Medium Minor / Informative Pass

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

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1129
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) public onlyOwner {
  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#L1124
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.

BT - Burns Tokens

Criticality	Critical
Location	contracts/CLRS.sol#L1336
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.mul(percentForLPBurn).div(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
●	AFI	Accumulated Fees Inconsistency	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
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

AFI - Accumulated Fees Inconsistency

Criticality	Critical
Location	contracts/CLRS.sol#L1279
Status	Unresolved

Description

The contract has a `rescueAnyBEP20Tokens` function that withdraws tokens from the contract. However, the contract does not update the variable that tracks the accumulated fees when the withdrawal occurs. This can result in incorrect fee calculations and cause the swap function to revert.

```
function swapBack() private {
    uint256 contractBalance = balanceOf(address(this));
    uint256 totalTokensToSwap = tokensForLiquidity + tokensForMarketing
+ tokensForDev + tokensForCharity;
    ...
}
```

Recommendation

It is recommended to update the variables that tracks the accumulated fees every time a token withdrawal occurs.

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.

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Calories.sol#L896,897
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.

```
uint256 public manualBurnFrequency = 30 minutes
uint256 public lastManualLpBurnTime
```

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	Calories.sol#L50,51,68,741,928,940,942,944,1065,1074,1125,1324
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);
mapping (address => bool) public _isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address indexed
oldWallet);
event devWalletUpdated(address indexed newWallet, address indexed
oldWallet);
event charityWalletUpdated(address indexed newWallet, address indexed
oldWallet);
uint256 _liquidityFee
uint256 _devFee
uint256 _buyCharityFee
uint256 _marketingFee
uint256 _sellCharityFee
uint _amount

...
```

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#L1042,1048,1053,1066,1075,1327
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
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#L1326
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#L1217,1218,1219,1220,1221,1225,1226,1227,1228,1229
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.

```
fees = amount.mul(sellTotalFees).div(100)
tokensForDev += fees * sellDevFee / sellTotalFees
```

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#L977
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Calories.sol#L1102,1107
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.

```
marketingWallet = newMarketingWallet
devWallet = 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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Calories.sol#L1126
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		

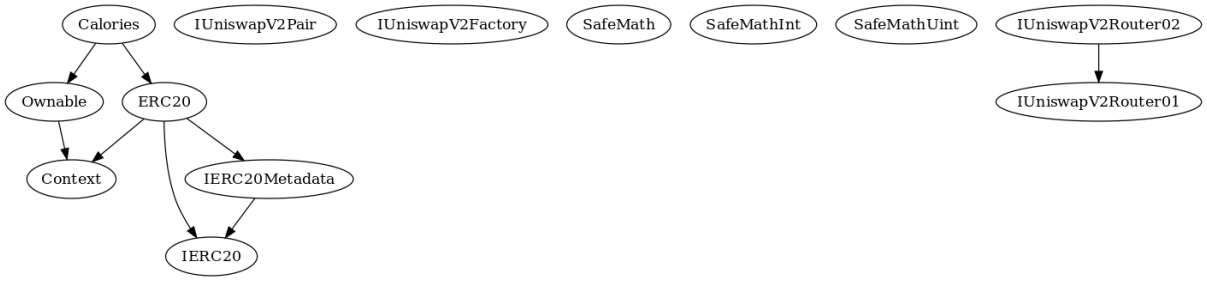
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	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		

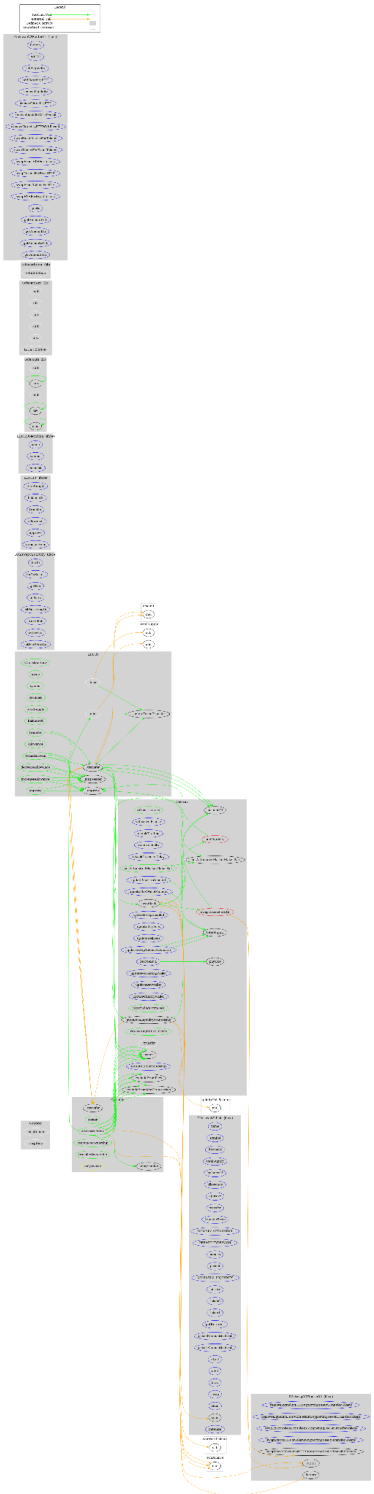
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	updateCharityWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	rescueBNB	External	✓	onlyOwner
	rescueAnyBEP20Tokens	Public	✓	onlyOwner
	_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 draining the contract's tokens and transferring funds to the team's wallet. 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

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

Cyberscope

Audit Report

ObeseFans Calories

June 2023

Network	BSC
Address	0x69830556232e56a6a33672ad6b8e5937d22ce90c
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 4
Findings Breakdown **5**
Analysis **6**
BT - Burns Tokens 7
Description 7
Recommendation 7
Diagnostics **9**
OCTD - Transfers Contract's Tokens 10
Description 10
Recommendation 10
ULTW - Transfers Liquidity to Team Wallet 12
Description 12
Recommendation 12
RE - Redundant Event 13
Description 13
Recommendation 13
RSML - Redundant SafeMath Library 14
Description 14
Recommendation 14
L04 - Conformance to Solidity Naming Conventions 15
Description 15
Recommendation 16
L05 - Unused State Variable 17
Description 17
Recommendation 17
L07 - Missing Events Arithmetic 18
Description 18
Recommendation 18
L08 - Tautology or Contradiction 19
Description 19
Recommendation 19
L09 - Dead Code Elimination 20
Description 20
Recommendation 20
L13 - Divide before Multiply Operation 22
Description 22

Recommendation 22
L15 - Local Scope Variable Shadowing 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

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

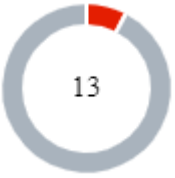
Audit Updates

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	✓	-

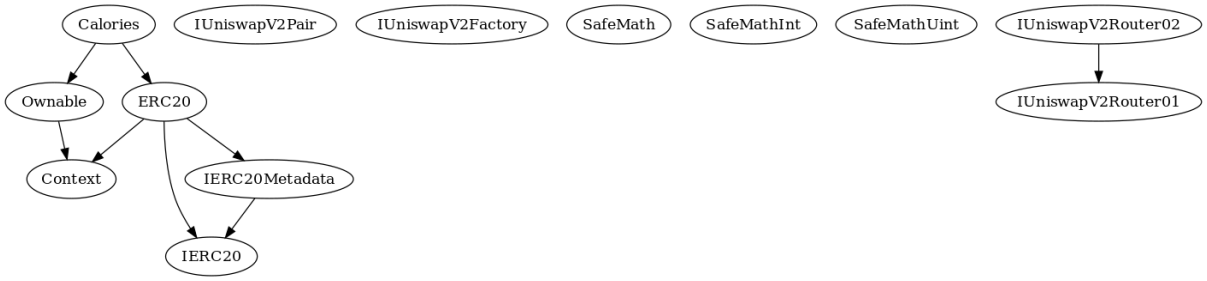
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	Interface			
	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	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Calories	Implementation	ERC20, Ownable		
		Public	✓	ERC20

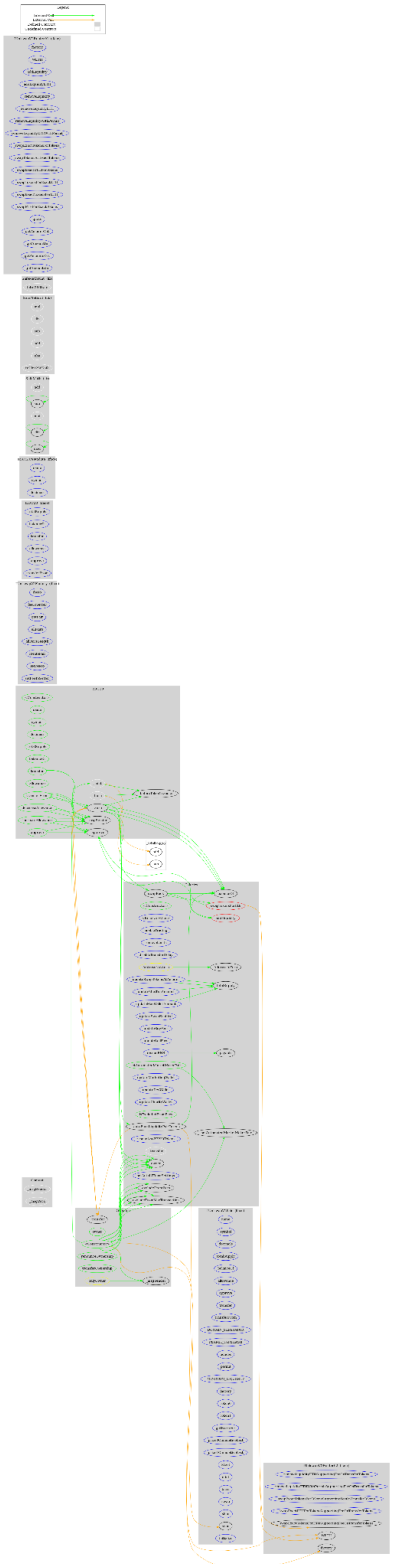
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	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 AddressNotZero
	_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

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

Cyberscope

Audit Report

Clrs NFT

June 2023

Network	BSC
Address	0x50704d2c0416f52af7b922cf264e55efd5065da2
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 2
Introduction 3
Findings Breakdown 4
Diagnostics 5
L02 - State Variables could be Declared Constant 6
Description 6
Recommendation 6
L04 - Conformance to Solidity Naming Conventions 7
Description 7
Recommendation 8
L09 - Dead Code Elimination 9
Description 9
Recommendation 9
L13 - Divide before Multiply Operation 11
Description 11
Recommendation 11
L15 - Local Scope Variable Shadowing 12
Description 12
Recommendation 12
L17 - Usage of Solidity Assembly 13
Description 13
Recommendation 13
Functions Analysis 14
Inheritance Graph 22
Flow Graph 23
Summary 24
Disclaimer 25
About Cyberscope 26

Review

Contract Name	ObeseFansVillainsFood
Deploy	https://bscscan.com/address/0x50704d2c0416f52af7b922cf264e55efd5065da2
Symbol	OBFVILLAINS

Audit Updates

Initial Audit	18 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v1/nft.pdf
Corrected Phase 2	12 May 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v2/nft.pdf
Corrected Phase 3	13 Jun 2023

Source Files

Filename	SHA256
ObeseFansVillainsFood.sol	0000d849c9307405c61db63316bd6d2f021a06e45206ee2376bf62ab6b991445

Introduction

This is a contract for an NFT presale. The NFTs are ERC721 tokens.

The contract owner can configure

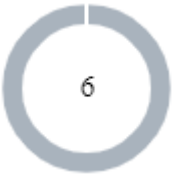
- The minting cost
- The maximum number of NFTs that can be minted per session.
- The NFTs URIs, the base URI extension.
- The whitelisted addresses.
- The mint limit per address.
- The base URIs.
- The base extension of the NFTs URI.

The contract owner has the authority to

- Reveal the not revealed URI.
- Withdraw funds that send 5% of the balance to HashLips address, and the remainder to the contract owner.

Contract users have to pay the mint costs except for the owner who is exempt.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NFTPresale.sol#L1678
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.

```
ess dev = 0x002B2eAD67fE0c3dd4E6eCCB035759848D8D6768;
```

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	NFTPresale.sol#L1652,1702,1726,1766,1770,1774,1779,1783,1787,1791,1795
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.

```
DOMAIN_SEPARATOR
_mintAmount
_owner

_limit
_newCost
_newmaxMintAmount
_newBaseURL
_newBaseExtension
_notRevealedURI
_state
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	NFTPresale.sol#L261,271,290,304,321,331,346,356,371,395,407,442,449,457,468,478,563,581,617,628,670,719,732,762,802,811,826,926,1142,1324,1563,1567,1571,1579,1584,1593,1609
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 functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0,
errorMessage);
    }

...
```

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	NFTPresale.sol#L525,528,540,544,545,546,547,548,549,555
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
prod0 := div(prod0, twos)
result = prod0 * inverse
```

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	NFTPresale.sol#L1687,1688,1726
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.

```
_name,_symbol, _owner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	NFTPresale.sol#L412,489,783,1264
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }


...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC721Metadata	Interface	IERC721		

	name	External		-
	symbol	External		-
	tokenURI	External		-
Address	Library			
	isContract	Internal		
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Math	Library			

	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-

ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	

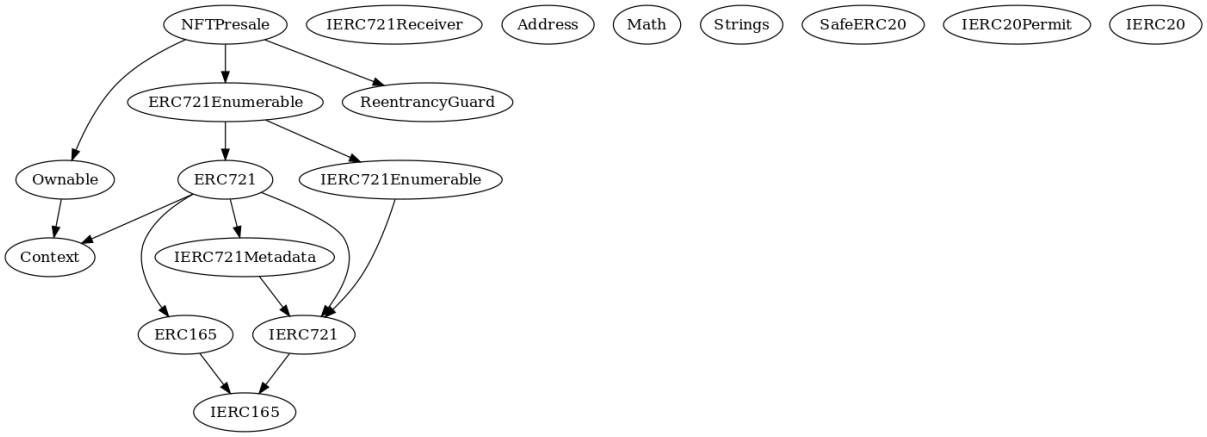
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	_unsafeIncreaseBalance	Internal	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	

	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			

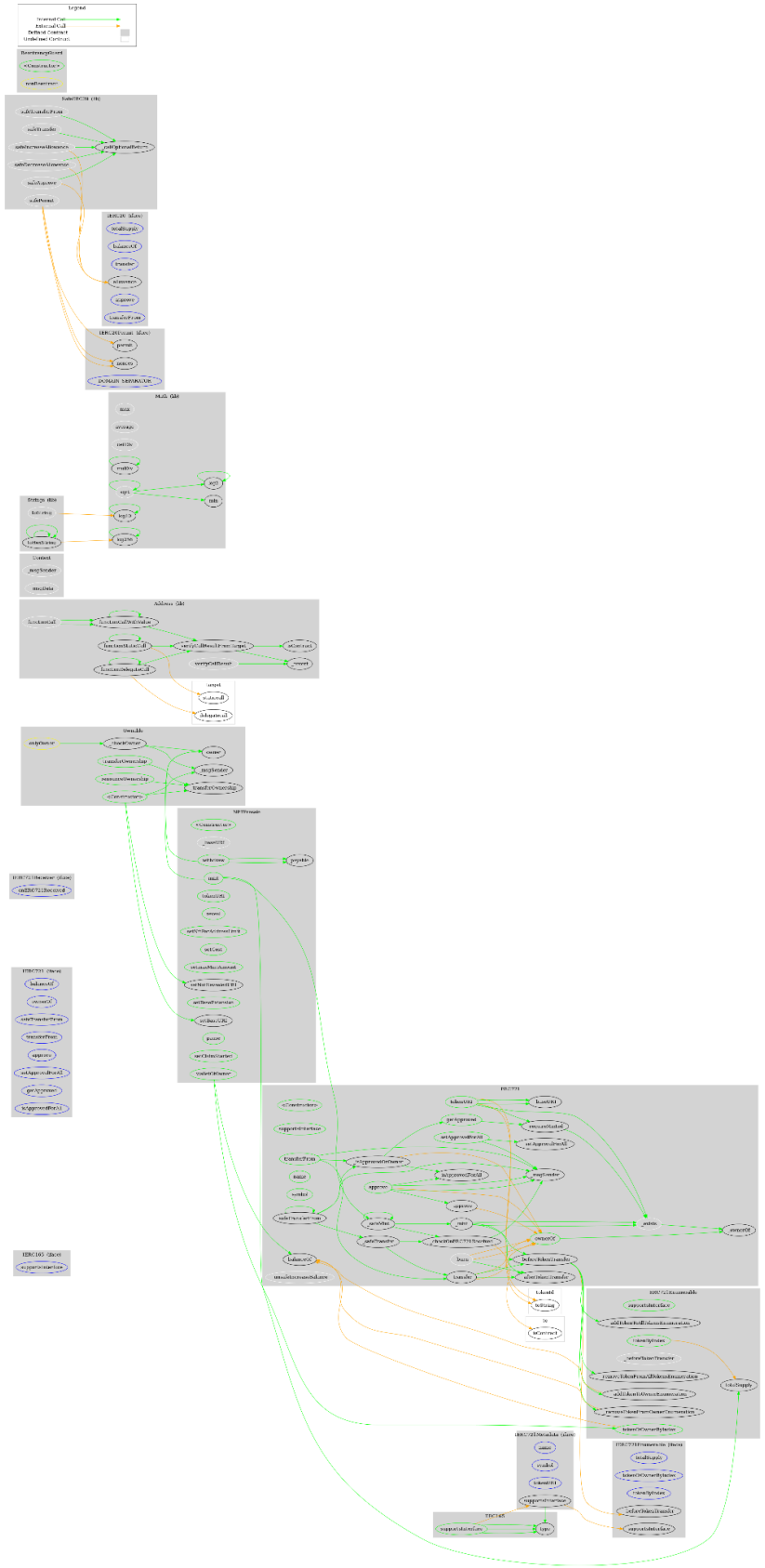
		Public	✓	-
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
NFTPresale	Implementation	ERC721Enumerable, Ownable, ReentrancyGuard		
		Public	✓	ERC721
	_baseURI	Internal		
	mint	Public	Payable	nonReentrant
	walletOfOwner	Public		-
	tokenURI	Public		-
	reveal	Public	✓	onlyOwner

	setNftPerAddressLimit	Public	✓	onlyOwner
	setCost	Public	✓	onlyOwner
	setmaxMintAmount	Public	✓	onlyOwner
	setBaseURI	Public	✓	onlyOwner
	setBaseExtension	Public	✓	onlyOwner
	setNotRevealedURI	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	setClaimStarted	Public	✓	onlyOwner
	withdraw	Public	Payable	onlyOwner nonReentrant

Inheritance Graph



Flow Graph



Summary

ObeseFans Calories contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Clrs Presale

June 2023

Network	BSC
Address	0xdE78Da0aFeC5808F187d7f4B793dde63BA14Ec3E
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Functionality 5
Findings Breakdown **6**
Diagnostics **7**
PTD - Presale Token Drain 8
Description 8
Recommendation 8
MSE - Missing Solidity Events 9
Description 9
Recommendation 9
IDI - Immutable Declaration Improvement 10
Description 10
Recommendation 10
L04 - Conformance to Solidity Naming Conventions 11
Description 11
Recommendation 11
L07 - Missing Events Arithmetic 13
Description 13
Recommendation 13
L09 - Dead Code Elimination 14
Description 14
Recommendation 15
L16 - Validate Variable Setters 16
Description 16
Recommendation 16
L17 - Usage of Solidity Assembly 17
Description 17
Recommendation 17
Functions Analysis **18**
Inheritance Graph **22**
Flow Graph **23**
Summary **24**
Disclaimer **25**
About Cyberscope **26**

Review

Explorer	https://bscscan.com/address/0xde78da0afec5808f187d7f4b793dde63ba14ec3e

Audit Updates

Initial Audit	18 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v1/presale.pdf
Corrected Phase 2	12 May 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v2/presale.pdf
Corrected Phase 3	13 Jun 2023

Source Files

Filename	SHA256
CLRSPresale.sol	fdeb7ab75d69c719b25d5cff6752906bb2a44af711a7a9b9f4106cadab3f7da5

Introduction

The CLRSPresale smart contract implements a presale contract for a token.

The contract defines several state variables

- `_contributions`, which is a mapping of addresses to the amount of ether contributed by each address; _token, which is the token being sold.
- `_tokenDecimals`, which is the number of decimals of the token; _wallet, which is the address that will receive the funds raised; _rate, which is the number of tokens that can be purchased for each wei of ether;
- `_weiRaised`, which is the total amount of wei raised.
- `endICO`, which is the end time of the pre-sale event.
- `minPurchase`, which is the minimum amount of ether that can be purchased by a user.
- `maxPurchase`, which is the maximum amount of ether that can be contributed by an address.
- `hardCap`, which is the maximum amount of ether that can be raised.
- `softCap`, which is the minimum amount of ether that needs to be raised.
- `availableTokensICO`, which is the number of tokens available for sale during the pre-sale event.
- `startRefund`, which is a boolean flag indicating whether a refund process has started.
- `refundStartDate`, which is the start time of the refund process.

Functionality

The `startICO` function is used to start the pre-sale event with the necessary arguments.

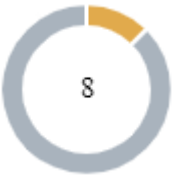
The `stopICO` function is used to stop the pre-sale event. If the soft cap has been reached, the funds raised are transferred to the wallet address. Otherwise, the `startRefund` flag is set to true, and the `refundStartDate` is set to the current time.

The `buyTokens` function is used to purchase tokens during the pre-sale event. It performs the necessary check to validate that the purchase is valid. Then the number of tokens purchased is calculated based on the `_rate` and `_tokenDecimals` variables, the necessary updates are taking place, and the `TokensPurchased` event is emitted.

The `claimTokens` function is used to claim tokens after the pre-sale event has ended. If the presale is in the correct valid state, the number of tokens purchased by the sender is calculated. The necessary state variables are updated. Finally, the tokens are transferred to the sender, and the `TokensClaimed` event is emitted.

The `withdraw` function is accessible by the owner to withdraw the ether raised during the pre-sale event. If the startRefund flag is false, and either the refund period has ended or no refunds have been requested, the ethers are transferred to the wallet address, and 5% of the ether are transferred to a specific address.

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	PTD	Presale Token Drain	Unresolved
Minor / Informative	MSE	Missing Solidity Events	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved

PTD - Presale Token Drain

Criticality	Medium
Location	contracts/CLRSPresale.sol#L452
Status	Unresolved

Description

The contract currently allows the owner to withdraw tokens from the presale contract, which can lead to a reduction in the amount of tokens available for sale during the presale period. This can result in a loss of funds for investors who have already purchased tokens during the presale.

```
function takeTokens(IERC20 tokenAddress)  public onlyOwner
icoNotActive {
    IERC20 tokenBEP = tokenAddress;
    uint256 tokenAmt = tokenBEP.balanceOf(address(this));
    require(tokenAmt > 0, 'BEP-20 balance is 0');
    tokenBEP.safeTransfer(_wallet, tokenAmt);
}
```

Recommendation

It is recommended to remove functionality that allows the contract owner to withdraw tokens from the presale contract. The presale contract should only allow for token purchases by investors and the contract owner should not have the ability to interfere with the presale process.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L400,421,461
Status	Unresolved

Description

The contract has some actions that a user can take on the contract that do not emit any 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.

```
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp);
require(_wallet != address(0));
require(msg.sender != address(0));
```

Recommendation

It is recommended adding events to the code that are emitted whenever a user interacts with the contract in a significant way. These events should include the 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CLRSPresale.sol#L311,312
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.

```
_token
_tokenDecimals
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CLRSPresale.sol#L260,279,281,283,284,285,328
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);
mapping (address => uint256) public _contributions
IERC20 public _token
address payable public _wallet
uint256 public _rate
uint256 public _weiRaised
uint _hardCap
uint _maxPurchase
uint _minPurchase
uint _softCap
```

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	CLRSPresale.sol#L337
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.

```
endICO = endDate
```

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	CLRSPresale.sol#L91,98,110,125,129,138,142,167,198,202,210,215,224
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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	CLRSPresale.sol#L419
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.

```
_wallet = 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	CLRSPresale.sol#L181
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 {
                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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
		Public	✓	-
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	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	

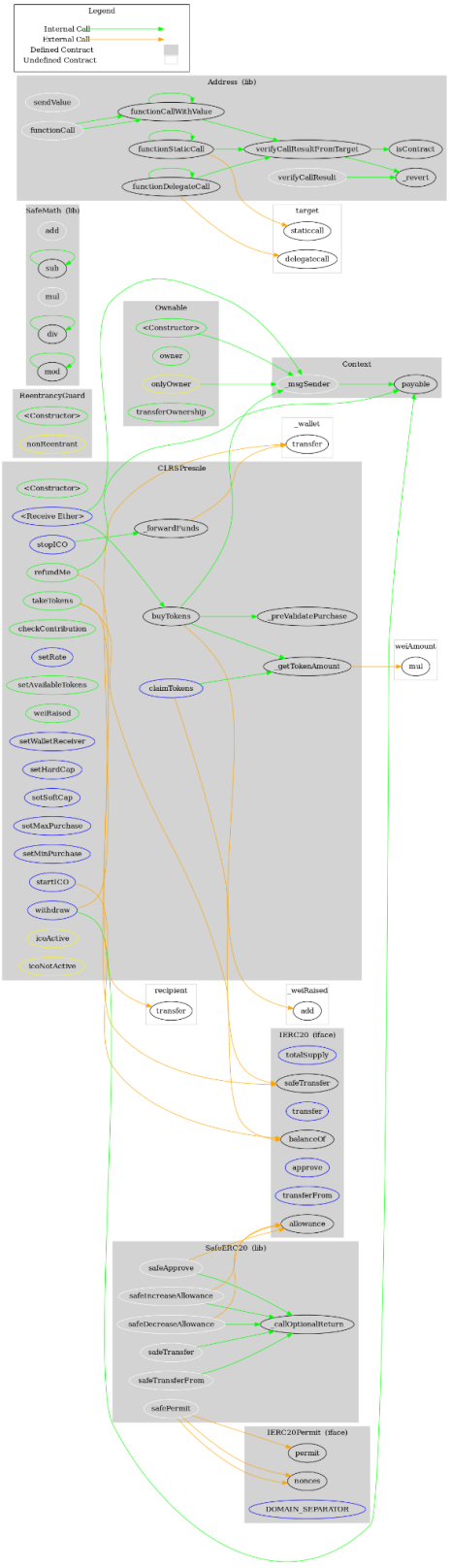
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
CLRSPresale	Implementation	ReentrancyGuard, Context, Ownable		
		Public	✓	-
		External	Payable	-
	startICO	External	✓	onlyOwner icoNotActive
	stopICO	External	✓	onlyOwner icoActive
	buyTokens	Public	Payable	nonReentrant icoActive

	_preValidatePurchase	Internal		
	claimTokens	External	✓	icoNotActive nonReentrant
	_getTokenAmount	Internal		
	_forwardFunds	Internal	✓	
	withdraw	External	✓	onlyOwner icoNotActive
	checkContribution	Public		-
	setRate	External	✓	onlyOwner icoNotActive
	setAvailableTokens	Public	✓	onlyOwner icoNotActive
	weiRaised	Public		-
	setWalletReceiver	External	✓	onlyOwner
	setHardCap	External	✓	onlyOwner
	setSoftCap	External	✓	onlyOwner
	setMaxPurchase	External	✓	onlyOwner
	setMinPurchase	External	✓	onlyOwner
	takeTokens	Public	✓	onlyOwner icoNotActive
	refundMe	Public	✓	icoNotActive nonReentrant

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
SafeMath (lib)
add
sub
mul
div
mod
Ownable
<Constructor>
owner
onlyOwner
transferOwnership
Context
_msgSender
payable
ReentrancyGuard
<Constructor>
nonReentrant
_wallet
transfer
CLRSPresale
<Constructor>
<Receive Ether>
forwardFunds
stopICO
refundMe
takeTokens
checkContribution
setRate
setAvailableTokens
weiRaised
setWalletReceiver
setHardCap
setSoftCap
setMaxPurchase
setMinPurchase
startICO
withdraw
icoActive
icoNotActive
buyTokens
preValidatePurchase
getTokenAmount
claimTokens
weiAmount
mul
recipient
transfer
_weiRaised
add
IERC20 (iface)
totalSupply
safeTransfer
transfer
balanceOf
approve
transferFrom
allowance
SafeERC20 (lib)
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
callOptionalReturn
safeTransfer
safeTransferFrom
safePermit
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR

Summary

ObeseFans Calories contract implements a presale mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

ObeseFans Calories

April 2023

SHA256 2f3a940b2aeeb3d828af9e62604f65f7eb44444ac4acacb8612a0d7f1243d836

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Findings Breakdown **4**
Analysis **5**
OCTD - Transfers Contract's Tokens 6
Description 6
Recommendation 6
ULTW - Transfers Liquidity to Team Wallet 7
Description 7
Recommendation 7
BT - Burns Tokens 8
Description 8
Recommendation 9
Diagnostics **10**
AFI - Accumulated Fees Inconsistency 11
Description 11
Recommendation 11
RSML - Redundant SafeMath Library 12
Description 12
Recommendation 12
L02 - State Variables could be Declared Constant 13
Description 13
Recommendation 13
L04 - Conformance to Solidity Naming Conventions 14
Description 14
Recommendation 15
L05 - Unused State Variable 16
Description 16
Recommendation 16
L07 - Missing Events Arithmetic 17
Description 17
Recommendation 17
L08 - Tautology or Contradiction 18
Description 18
Recommendation 18
L09 - Dead Code Elimination 19
Description 19

Recommendation 19
L13 - Divide before Multiply Operation 21
Description 21
Recommendation 21
L15 - Local Scope Variable Shadowing 22
Description 22
Recommendation 22
L16 - Validate Variable Setters 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Contract Name	Calories
Testing Deploy	https://testnet.bscscan.com/address/0x4ca6812c88db2c319c20bfabc0cfa53f7cd5921f
Symbol	$CLRS
Decimals	18
Total Supply	10.000.000.000

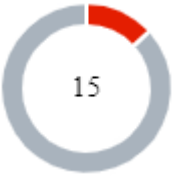
Audit Updates

Initial Audit	11 Apr 2023

Source Files

Filename	SHA256
contracts/CLRS.sol	2f3a940b2aeeb3d828af9e62604f65f7eb44444ac4acacb8612a0d7f1243d836

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Analysis

Critical Medium Minor / Informative Pass

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

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1123
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) public onlyOwner {
  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
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.

BT - Burns Tokens

Criticality	Critical
Location	contracts/CLRS.sol#L1330
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.mul(percentForLPBurn).div(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
●	AFI	Accumulated Fees Inconsistency	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
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

AFI - Accumulated Fees Inconsistency

Criticality	Critical
Location	contracts/CLRS.sol#L1279
Status	Unresolved

Description

The contract has a `rescueAnyBEP20Tokens` function that withdraws tokens from the contract. However, the contract does not update the variable that tracks the accumulated fees when the withdrawal occurs. This can result in incorrect fee calculations and cause the swap function to revert.

```
function swapBack() private {
    uint256 contractBalance = balanceOf(address(this));
    uint256 totalTokensToSwap = tokensForLiquidity + tokensForMarketing
+ tokensForDev + tokensForCharity;
    ...
}
```

Recommendation

It is recommended to update the variables that tracks the accumulated fees every time a token withdrawal occurs.

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/CLRS.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/CLRS.sol#L893,894
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.

```
uint256 public manualBurnFrequency = 30 minutes
uint256 public lastManualLpBurnTime
```

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/CLRS.sol#L47,48,65,738,876,925,937,939,941,1063,1072,1123,1322
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);
address public constant deadAddress = address(0xdead)
mapping (address => bool) public
_isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event devWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event charityWalletUpdated(address indexed newWallet, address
indexed oldWallet);
uint256 _liquidityFee
uint256 _devFee
uint256 _buyCharityFee
uint256 _marketingFee
uint256 _sellCharityFee

...
```

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/CLRS.sol#L669
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	contracts/CLRS.sol#L1040,1046,1051,1064,1073,1325
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
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	contracts/CLRS.sol#L1324
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	contracts/CLRS.sol#L414,715,721,728
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	contracts/CLRS.sol#L1215,1216,1217,1218,1219,1223,1224,1225,1226,1227
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.

```
tokensForMarketing += fees * sellMarketingFee / sellTotalFees
fees = amount.mul(buyTotalFees).div(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	contracts/CLRS.sol#L974
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1100,1105
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.

```
marketingWallet = newMarketingWallet
devWallet = 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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1124
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		

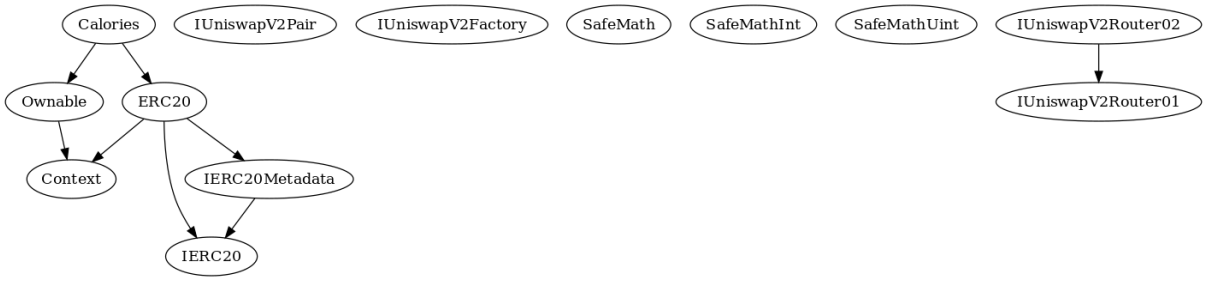
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	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		

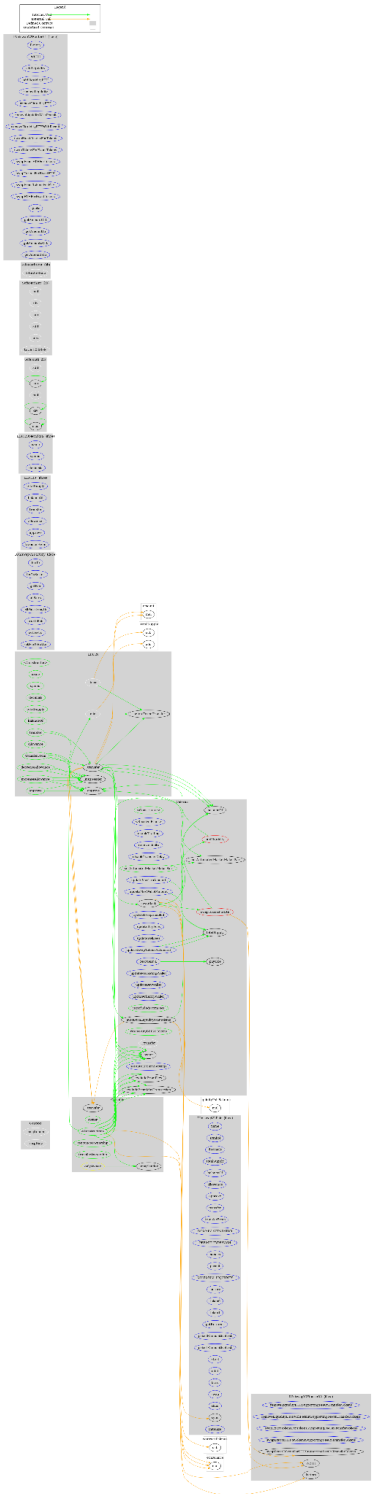
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	updateCharityWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	rescueBNB	External	✓	onlyOwner
	rescueAnyBEP20Tokens	Public	✓	onlyOwner
	_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 draining the contract's tokens and transferring funds to the team's wallet. 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

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

Cyberscope

Audit Report

Clrs NFT

April 2023

SHA256 14a69f641a693a0ebbc000c82cbe005451a3bc7144160afd72c04dfff8babf70

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Findings Breakdown **5**
Diagnostics **6**
FSO - For Statement Optimization 7
Description 7
Recommendation 7
MSC - Missing Sanity Check 8
Description 8
Recommendation 8
DSM - Data Structure Misuse 9
Description 9
Recommendation 9
RSML - Redundant SafeMath Library 10
Description 10
Recommendation 10
L02 - State Variables could be Declared Constant 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 12
L09 - Dead Code Elimination 14
Description 14
Recommendation 14
L11 - Unnecessary Boolean equality 16
Description 16
Recommendation 16
L14 - Uninitialized Variables in Local Scope 17
Description 17
Recommendation 17
L15 - Local Scope Variable Shadowing 18
Description 18
Recommendation 18
L17 - Usage of Solidity Assembly 19
Description 19

Recommendation 19
Functions Analysis **20**
Inheritance Graph **29**
Flow Graph **30**
Summary **31**
Disclaimer **32**
About Cyberscope **33**

Review

Contract Name	NFTPresale
Testing Deploy	https://testnet.bscscan.com/address/0x488b8c3a6c19df49bbf6807aefab1a255251e5f5
Symbol	TST

Audit Updates

Initial Audit	18 Apr 2023

Source Files

Filename	SHA256
contracts/NFTPresale.sol	14a69f641a693a0ebbc000c82cbe005451a3bc7144160afd72c04dfff8babf70

Introduction

This is a contract for an NFT presale. The NFTs are ERC721 tokens.

The contract owner can configure

- The minting cost
- The maximum number of NFTs that can be minted per session.
- The NFTs URIs, the base URI extension.
- The whitelisted addresses.
- The mint limit per address.
- The base URIs.
- The base extension of the NFTs URI.

The contract owner has the authority to

- Reveal the not revealed URI.
- Withdraw funds that send 5% of the balance to HashLips address, and the remainder to the contract owner.

Contract users have to pay the mint costs except for the owner who is exempt.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	FSO	For Statement Optimization	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	DSM	Data Structure Misuse	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

FSO - For Statement Optimization

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1774
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 contract is aggregating the user's minted balance inside a for loop that does not depend on the loop index since there is the variable `_mintAmount`. This means that the statement is being unnecessarily executed multiple times.

```
for (uint256 i = 1; i <= _mintAmount; i++) {
    addressMintedBalance[msg.sender]++;
    _safeMint(msg.sender, supply + i);
}
```

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.

The contract could move the user's minted balance aggregation after the for-loop.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1865,1837
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The whitelisted addresses can be set to zero addresses.

```
function whitelistUsers(address[] calldata _users) public
onlyOwner {
    delete whitelistedAddresses;
    whitelistedAddresses = _users;
}
```

The max mint amount should be greater than 0 since the min mint amount is greater than zero. `require(_mintAmount > 0, "need to mint at least 1 NFT");`

```
function setmaxMintAmount(uint256 _newmaxMintAmount) public
onlyOwner {
    maxMintAmount = _newmaxMintAmount;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications. The argument address should not be set to zero addresses and the ma mint amount should be greater than 0.

DSM - Data Structure Misuse

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1780
Status	Unresolved

Description

The contract uses the valuable `whitelistedAddresses` as an array. The business logic of the contract does not require to iterate this structure sequentially. Thus, unnecessary loops are produced that increase the required gas.

```
function isWhitelisted(address _user) public view returns
(bool) {
    for (uint i = 0; i < whitelistedAddresses.length; i++) {
      if (whitelistedAddresses[i] == _user) {
          return true;
      }
    }
    return false;
}
```

Recommendation

The contract could use a data structure that provides instant access. For instance, a Set or a Map would fit better to the business logic of the contract. This way the time complexity will be reduced from o(n) to o(1).

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/NFTPresale.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/NFTPresale.sol#L1732
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.

```
public maxSupply = 2500;
```

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/NFTPresale.sol#L1381,1709,1758,1780,1789,1829,1833,1837,1841,1845,1849,1853,1857,1861,1865
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.

```
_mintAmount
_user

_owner


...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L293,318,328,347,361,378,388,403,413,428,452,464,499,506,514,525,535,620,638,674,685,727,776,789,819,859,868,883,983,1199,1381,1620,1624,1628,1636,1641,1650,1666
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1766,1814
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
if(Whitelisted == true) {

if(revealed == false) {
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1796
Status	Unresolved

Description

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

```
uint256 i
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1743,1744,1789
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.

```
string memory _name,

string memory _symbol,

address _owner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L469,546,840,1321
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }


...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-

	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-
	tokenURI	External		-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		

	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		

	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-

	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	__unsafe_increaseBalance	Internal	✓	

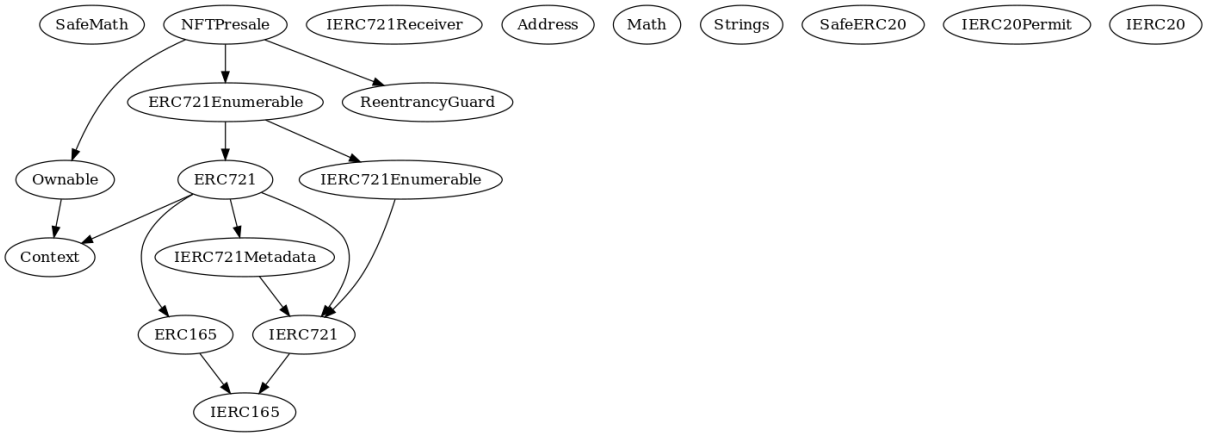
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	
	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

	_transferOwnership	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			
		Public	✓	-
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

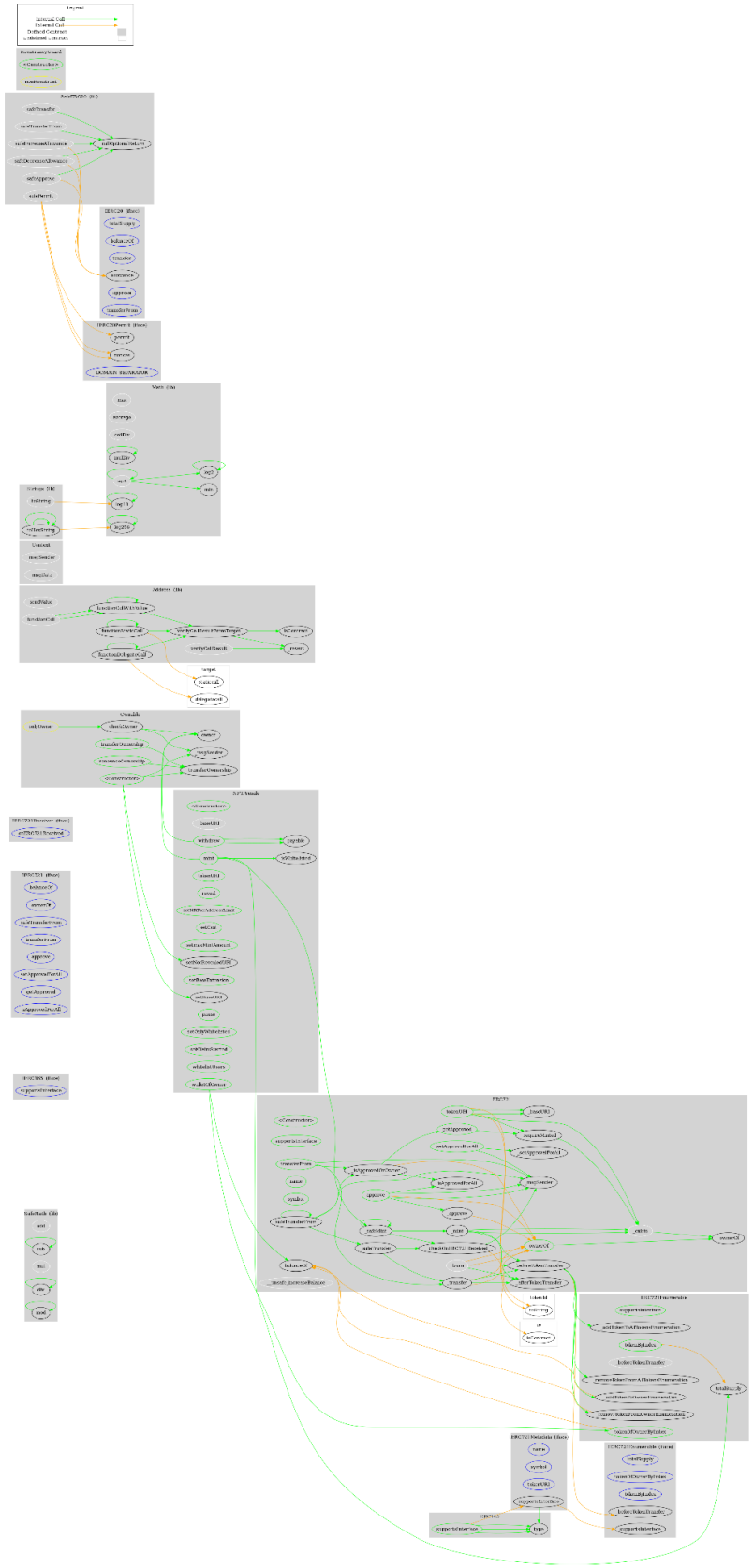
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
NFTPresale	Implementation	ERC721Enumerable, Ownable, ReentrancyGuard		
		Public	✓	ERC721
	_baseURI	Internal		
	mint	Public	Payable	nonReentrant
	isWhitelisted	Public		-
	walletOfOwner	Public		-
	tokenURI	Public		-
	reveal	Public	✓	onlyOwner
	setNftPerAddressLimit	Public	✓	onlyOwner
	setCost	Public	✓	onlyOwner
	setmaxMintAmount	Public	✓	onlyOwner
	setBaseURI	Public	✓	onlyOwner
	setBaseExtension	Public	✓	onlyOwner
	setNotRevealedURI	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	setOnlyWhitelisted	Public	✓	onlyOwner
	setClaimStarted	Public	✓	onlyOwner
	whitelistUsers	Public	✓	onlyOwner

	withdraw	Public	Payable	onlyOwner nonReentrant

Inheritance Graph



Flow Graph



Summary

ObeseFans Calories contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Clrs Presale

April 2023

SHA256 a70c5e57f04cd95793386cdca7a5ce148b6751d9e7d98983ccefe1a55146b880

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Functionality 4
Findings Breakdown **6**
Diagnostics **7**
PMI - Presale Mechanism Inconsistency 9
Description 9
Recommendation 9
MSC - Missing Sanity Check 10
Description 10
Recommendation 11
PTD - Presale Token Drain 12
Description 12
Recommendation 12
MPPC - Missing Purchase Prevalidation Check 13
Description 13
Recommendation 14
VO - Variable Optimization 15
Description 15
Recommendation 15
CI - Calculation Inconsistency 16
Description 16
Recommendation 16
MRPC - Missing Refund Prevalidation Check 17
Description 17
Recommendation 17
MSE - Missing Solidity Events 18
Description 18
Recommendation 18
RSML - Redundant SafeMath Library 19
Description 19
Recommendation 19
IDI - Immutable Declaration Improvement 20
Description 20
Recommendation 20
L04 - Conformance to Solidity Naming Conventions 21

Description 21
Recommendation 21
L07 - Missing Events Arithmetic 23
Description 23
Recommendation 23
L09 - Dead Code Elimination 24
Description 24
Recommendation 25
L11 - Unnecessary Boolean equality 26
Description 26
Recommendation 26
L16 - Validate Variable Setters 27
Description 27
Recommendation 27
L17 - Usage of Solidity Assembly 28
Description 28
Recommendation 28
Functions Analysis **29**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Testing Deploy	https://testnet.bscscan.com/address/0x2b675b486a2381384abfb41ee135d06d9093e628

Audit Updates

Initial Audit	18 Apr 2023

Source Files

Filename	SHA256
contracts/CLRSPresale.sol	a70c5e57f04cd95793386cdca7a5ce148b6751d9e7d98983ccefe1a55146b880

Introduction

The CLRSPresale smart contract implements a presale contract for a token.

The contract defines several state variables

- `_contributions`, which is a mapping of addresses to the amount of ether contributed by each address; _token, which is the token being sold.
- `_tokenDecimals`, which is the number of decimals of the token; _wallet, which is the address that will receive the funds raised; _rate, which is the number of tokens that can be purchased for each wei of ether;
- `_weiRaised`, which is the total amount of wei raised.
- `endICO`, which is the end time of the pre-sale event.
- `minPurchase`, which is the minimum amount of ether that can be purchased by a user.
- `maxPurchase`, which is the maximum amount of ether that can be contributed by an address.
- `hardCap`, which is the maximum amount of ether that can be raised.
- `softCap`, which is the minimum amount of ether that needs to be raised.
- `availableTokensICO`, which is the number of tokens available for sale during the pre-sale event.
- `startRefund`, which is a boolean flag indicating whether a refund process has started.
- `refundStartDate`, which is the start time of the refund process.

Functionality

The `startICO` function is used to start the pre-sale event with the necessary arguments.

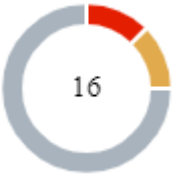
The `stopICO` function is used to stop the pre-sale event. If the soft cap has been reached, the funds raised are transferred to the wallet address. Otherwise, the `startRefund` flag is set to true, and the `refundStartDate` is set to the current time.

The `buyTokens` function is used to purchase tokens during the pre-sale event. It performs the necessary check to validate that the purchase is valid. Then the number of tokens purchased is calculated based on the `_rate` and `_tokenDecimals` variables, the necessary updates are taking place, and the `TokensPurchased` event is emitted.

The `claimTokens` function is used to claim tokens after the pre-sale event has ended. If the presale is in the correct valid state, the number of tokens purchased by the sender is calculated. The necessary state variables are updated. Finally, the tokens are transferred to the sender, and the `TokensClaimed` event is emitted.

The `withdraw` function is accessible by the owner to withdraw the ether raised during the pre-sale event. If the startRefund flag is false, and either the refund period has ended or no refunds have been requested, the ethers are transferred to the wallet address, and 5% of the ether are transferred to a specific address.

Findings Breakdown



Critical	2
Medium	2
Minor / Informative	12

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PMI	Presale Mechanism Inconsistency	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	PTD	Presale Token Drain	Unresolved
●	MPPC	Missing Purchase Prevalidation Check	Unresolved
●	VO	Variable Optimization	Unresolved
●	CI	Calculation Inconsistency	Unresolved
●	MRPC	Missing Refund Prevalidation Check	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

PMI - Presale Mechanism Inconsistency

Criticality	Critical
Location	contracts/CLRSPresale.sol#L461
Status	Unresolved

Description

The presale mechanism in the contract should have consistent parameters that are fixed and cannot be modified by the owner once the presale has started. All parameters, including the available token amount, should be set before the start of the presale to ensure consistency and fairness for all investors.

```
function setAvailableTokens(uint256 amount) public onlyOwner
icoNotActive{
    availableTokensICO = amount;
    emit AvailableTokenUpdated(amount);
}
```

Recommendation

It is recommended to have fixed `availableTokensICO` presale parameter before the start of the presale and prevent the owner from modifying it once the presale has started.

MSC - Missing Sanity Check

Criticality	Critical
Location	contracts/CLRSPresale.sol#L456,461,475,480,484
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The `_rate` variable could be set to zero and prevent users from taking tokens on the purchase transaction.

```
function setRate(uint256 newRate) external onlyOwner icoNotActive {
    _rate = newRate;
    emit RateUpdated(newRate);
}
```

The min purchase value can be set to values greater than the max purchase value and to zero.

```
function setMinPurchase(uint256 value) external onlyOwner{
    minPurchase = value;
    emit MinPurchaseUpdated(value);
}
```

The max purchase amount could be set to values lower than the min purchase amount.

```
function setMaxPurchase(uint256 value) external onlyOwner{
    maxPurchase = value;
    emit MaxPurchaseUpdated(value);
}
```

The hard cap could be set to values lower than the soft cap.

```
function setHardCap(uint256 value) external onlyOwner{
    hardCap = value;
    emit HardCapUpdated(value);
}
```

The soft value could be set to values greater than the hard cap.

```
function setSoftCap(uint256 value) external onlyOwner {
    softCap = value;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The `softCap` variable should exceed the hard cap.
- The `rate` variable should be greater than zero.
- The `maxPurchase` variable should be lower than or equal to `hardCap` .

The following requirements are used in the contract's implementation. The contract should re-apply the following requirements to the related setter functions.

- `require(_softCap < _hardCap, "Softcap must be lower than Hardcap");`
- `require(_minPurchase < _maxPurchase, "minPurchase must be lower than maxPurchase");`
- `require(_minPurchase > 0, '_minPurchase should > 0');`

PTD - Presale Token Drain

Criticality	Medium
Location	contracts/CLRSPresale.sol#L494
Status	Unresolved

Description

The contract currently allows the owner to withdraw tokens from the presale contract, which can lead to a reduction in the amount of tokens available for sale during the presale period. This can result in a loss of funds for investors who have already purchased tokens during the presale.

```
function takeTokens(IERC20 tokenAddress)  public onlyOwner
icoNotActive {
    IERC20 tokenBEP = tokenAddress;
    uint256 tokenAmt = tokenBEP.balanceOf(address(this));
    require(tokenAmt > 0, 'BEP-20 balance is 0');
    tokenBEP.safeTransfer(_wallet, tokenAmt);
}
```

Recommendation

It is recommended to remove functionality that allows the contract owner to withdraw tokens from the presale contract. The presale contract should only allow for token purchases by investors and the contract owner should not have the ability to interfere with the presale process.

MPPC - Missing Purchase Prevalidation Check

Criticality	Medium
Location	contracts/CLRSPresale.sol#L419
Status	Unresolved

Description

The contract allows users to buy tokens despite the issuance of a refund, which can lead to investors making invalid purchases and potential loss of funds.

```
function _preValidatePurchase(
    address beneficiary,
    uint256 weiAmount
  ) internal view {
    require(
      beneficiary != address(0),
      "Crowdsale: beneficiary is the zero address"
    );
    require(weiAmount != 0, "Crowdsale: weiAmount is 0");
    require(weiAmount >= minPurchase, "have to send at least:
minPurchase");
    require(
      _contributions[beneficiary].add(weiAmount) <=
maxPurchase,
      "can't buy more than: maxPurchase"
    );
    require((_weiRaised + weiAmount) <= hardCap, "Hard Cap
reached");
    this;
}
```

The contract allows the users to buy tokens even if there are not enough tokens for purchase.

```
function buyTokens(address beneficiary) public nonReentrant
icoActive payable {
  ...
  availableTokensICO = availableTokensICO - tokens;
  ...
}
```

Recommendation

It is recommended to add two extra pre-validation requirements in the token purchase function.

- Checking the `startRefund`, to prevent invalid purchases.
- Checking if there are enough available tokens for purchase.

VO - Variable Optimization

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L363
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 variable `tokenDecimals` is represented in `uint256`. The corresponding number will never exceed the value of 18, then the implementation of the contract does not necessarily require the use of a uint256 data type.

```
constructor (uint256 rate, address payable wallet, IERC20 token,
uint256 tokenDecimals)  {
    ...
    _tokenDecimals = 18 - tokenDecimals;
}
```

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.

The variable `tokenDecimals` could be represented to `uint8`.

CI - Calculation Inconsistency

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L413,414
Status	Unresolved

Description

The contract performs calculations using two different methods, either native calculations or the SafeMath library, which can make it difficult to read and understand. This inconsistency may also introduce errors or vulnerabilities in the contract's logic.

```
_weiRaised = _weiRaised.add(weiAmount);
availableTokensICO = availableTokensICO - tokens;
```

Recommendation

It is recommended to use native calculations since the contract is compiled using a version of Solidity greater than 0.8.0. The compiler performs calculation checks to prevent overflows and underflows, making it safe to use native calculations. This will make the code simpler and easier to read and understand.

MRPC - Missing Refund Prevalidation Check

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L501
Status	Unresolved

Description

The contract attempts to refund a user without verifying if the contract has enough balance to refund the user or if the user has any other contribution to refund.

```
function refundMe() public icoNotActive nonReentrant {
    require(startRefund == true, 'no refund available');
    require(msg.sender != address(0));
    uint amount = _contributions[msg.sender];
    if (address(this).balance >= amount) {
        _contributions[msg.sender] = 0;
        if (amount > 0) {
            address payable recipient = payable(msg.sender);
            recipient.transfer(amount);
            emit Refund(msg.sender, amount);
        }
    }
}
```

Recommendation

It is recommended to add pre-validation checks with require statements when attempting to refund a user. The checks should verify that the contract has enough balance to refund the user and that the user has contributions to refund. If either of these conditions are not met, the contract should revert with a clear error message indicating the reason for the revert.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L429,446,471,503
Status	Unresolved

Description

The contract has some actions that a user can take on the contract that do not emit any 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.

```
require(startRefund == false);
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp);
require(_wallet != address(0));
require(msg.sender != address(0));
```

Recommendation

It is recommended adding events to the code that are emitted whenever a user interacts with the contract in a significant way. 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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/CLRSPresale.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L362,363
Status	Unresolved

Description

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

```
_token
_tokenDecimals
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L310,330,332,334,335,336,379
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);
mapping (address => uint256) public _contributions
IERC20 public _token
address payable public _wallet
uint256 public _rate
uint256 public _weiRaised
uint _hardCap
uint _maxPurchase
uint _minPurchase
uint _softCap
```

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	contracts/CLRSPresale.sol#L388
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.

```
endICO = endDate
```

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/CLRSPresale.sol#L141,148,160,175,179,188,192,217,248,252,260,265,274
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L429,446,502
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(startRefund == false)
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp)
require(startRefund == true, 'no refund available')
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L472,509
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.

```
_wallet = newWallet
recipient.transfer(amount)
```

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/CLRSPresale.sol#L231
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 {
                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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
		Public	✓	-
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	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	

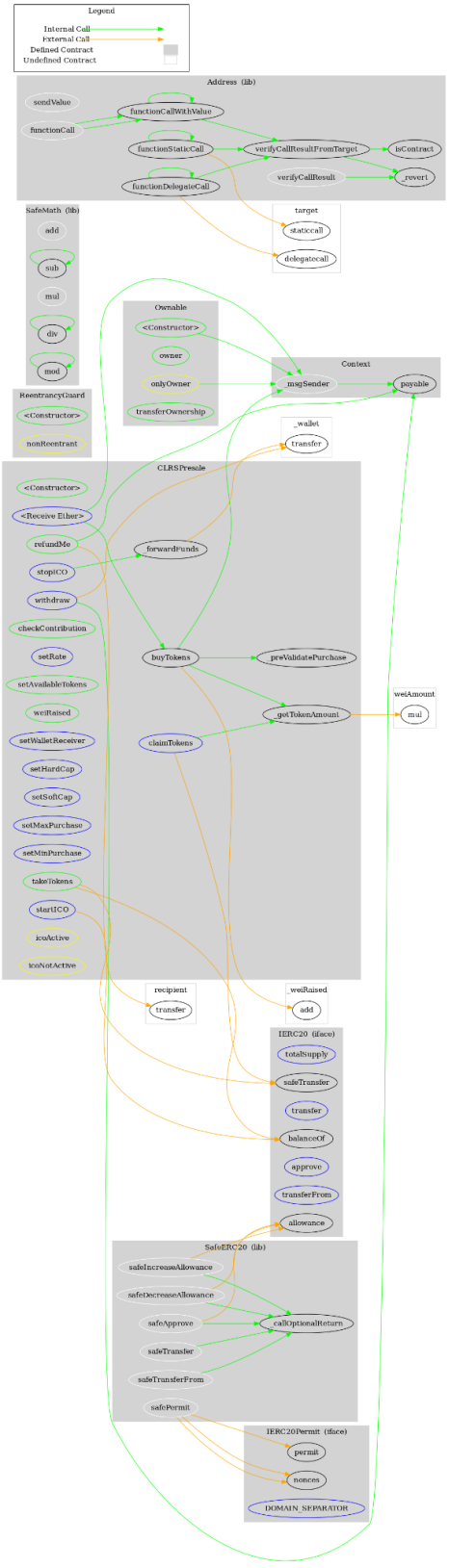
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
CLRSPresale	Implementation	ReentrancyGuard, Context, Ownable		
		Public	✓	-
		External	Payable	-
	startICO	External	✓	onlyOwner icoNotActive
	stopICO	External	✓	onlyOwner icoActive

	buyTokens	Public	Payable	nonReentrant icoActive
	_preValidatePurchase	Internal		
	claimTokens	External	✓	icoNotActive nonReentrant
	_getTokenAmount	Internal		
	_forwardFunds	Internal	✓	
	withdraw	External	✓	onlyOwner icoNotActive
	checkContribution	Public		-
	setRate	External	✓	onlyOwner icoNotActive
	setAvailableTokens	Public	✓	onlyOwner icoNotActive
	weiRaised	Public		-
	setWalletReceiver	External	✓	onlyOwner
	setHardCap	External	✓	onlyOwner
	setSoftCap	External	✓	onlyOwner
	setMaxPurchase	External	✓	onlyOwner
	setMinPurchase	External	✓	onlyOwner
	takeTokens	Public	✓	onlyOwner icoNotActive
	refundMe	Public	✓	icoNotActive nonReentrant

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
SafeMath (lib)
add
sub
mul
div
mod
Ownable
<Constructor>
owner
onlyOwner
transferOwnership
Context
msgSender
payable
ReentrancyGuard
<Constructor>
nonReentrant
_wallet
transfer
CLRSPresale
<Constructor>
<Receive Ether>
refundMe
stopICO
withdraw
checkContribution
setRate
setAvailableTokens
weiRaised
setWalletReceiver
setHardCap
setSoftCap
setMaxPurchase
setMinPurchase
takeTokens
startICO
icoActive
icoNotActive
forwardFunds
buyTokens
preValidatePurchase
_getTokenAmount
claimTokens
weiAmount
mul
recipient
transfer
weiRaised
add
IERC20 (iface)
totalSupply
safeTransfer
transfer
balanceOf
approve
transferFrom
allowance
SafeERC20 (lib)
safeIncreaseAllowance
safeDecreaseAllowance
safeApprove
safeTransfer
safeTransferFrom
safePermit
callOptionalReturn
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR

Summary

ObeseFans Calories contract implements a presale mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

ObeseFans Calories

April 2023

Network	BSC Testnet
Address	0xe94005A06fa7713d24ac992dDf01bD8D7e08DfCA
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Findings Breakdown **4**
Analysis **5**
OCTD - Transfers Contract's Tokens 6
Description 6
Recommendation 6
ULTW - Transfers Liquidity to Team Wallet 7
Description 7
Recommendation 7
BT - Burns Tokens 8
Description 8
Recommendation 9
Diagnostics **10**
AFI - Accumulated Fees Inconsistency 11
Description 11
Recommendation 11
RSML - Redundant SafeMath Library 12
Description 12
Recommendation 12
L02 - State Variables could be Declared Constant 13
Description 13
Recommendation 13
L04 - Conformance to Solidity Naming Conventions 14
Description 14
Recommendation 15
L05 - Unused State Variable 16
Description 16
Recommendation 16
L07 - Missing Events Arithmetic 17
Description 17
Recommendation 17
L08 - Tautology or Contradiction 18
Description 18
Recommendation 18
L09 - Dead Code Elimination 19
Description 19

Recommendation 19
L13 - Divide before Multiply Operation 21
Description 21
Recommendation 21
L15 - Local Scope Variable Shadowing 22
Description 22
Recommendation 22
L16 - Validate Variable Setters 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Explorer	https://testnet.bscscan.com/address/0xe94005a06fa7713d24ac992ddf01bd8d7e08dfca

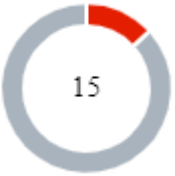
Audit Updates

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

Source Files

Filename	SHA256
Calories.sol	c9bcbb3edf12928f5098e98e3d5fe1da3b1a3eb3e311f0d0f432e7e4d03464ca

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Analysis

Critical Medium Minor / Informative Pass

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

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1129
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) public onlyOwner {
  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#L1124
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.

BT - Burns Tokens

Criticality	Critical
Location	contracts/CLRS.sol#L1336
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.mul(percentForLPBurn).div(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
●	AFI	Accumulated Fees Inconsistency	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
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

AFI - Accumulated Fees Inconsistency

Criticality	Critical
Location	contracts/CLRS.sol#L1279
Status	Unresolved

Description

The contract has a `rescueAnyBEP20Tokens` function that withdraws tokens from the contract. However, the contract does not update the variable that tracks the accumulated fees when the withdrawal occurs. This can result in incorrect fee calculations and cause the swap function to revert.

```
function swapBack() private {
    uint256 contractBalance = balanceOf(address(this));
    uint256 totalTokensToSwap = tokensForLiquidity + tokensForMarketing
+ tokensForDev + tokensForCharity;
    ...
}
```

Recommendation

It is recommended to update the variables that tracks the accumulated fees every time a token withdrawal occurs.

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.

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Calories.sol#L896,897
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.

```
uint256 public manualBurnFrequency = 30 minutes
uint256 public lastManualLpBurnTime
```

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	Calories.sol#L50,51,68,741,928,940,942,944,1065,1074,1125,1324
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);
mapping (address => bool) public _isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address indexed
oldWallet);
event devWalletUpdated(address indexed newWallet, address indexed
oldWallet);
event charityWalletUpdated(address indexed newWallet, address indexed
oldWallet);
uint256 _liquidityFee
uint256 _devFee
uint256 _buyCharityFee
uint256 _marketingFee
uint256 _sellCharityFee
uint _amount

...
```

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#L1042,1048,1053,1066,1075,1327
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
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#L1326
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#L1217,1218,1219,1220,1221,1225,1226,1227,1228,1229
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.

```
fees = amount.mul(sellTotalFees).div(100)
tokensForDev += fees * sellDevFee / sellTotalFees
```

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#L977
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Calories.sol#L1102,1107
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.

```
marketingWallet = newMarketingWallet
devWallet = 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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Calories.sol#L1126
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		

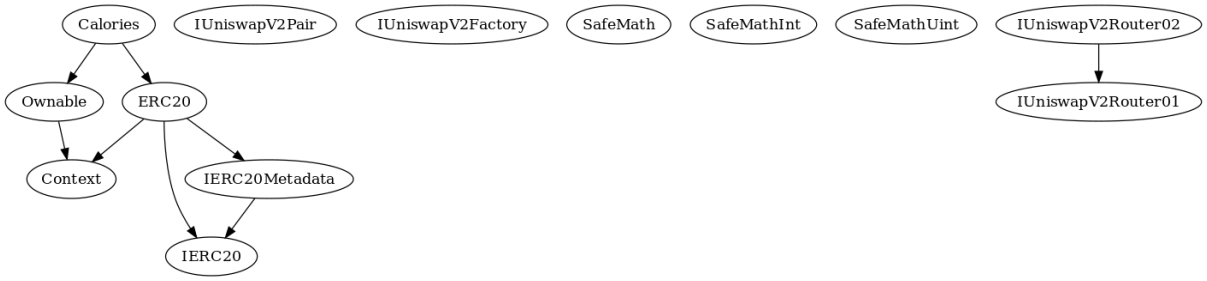
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	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		

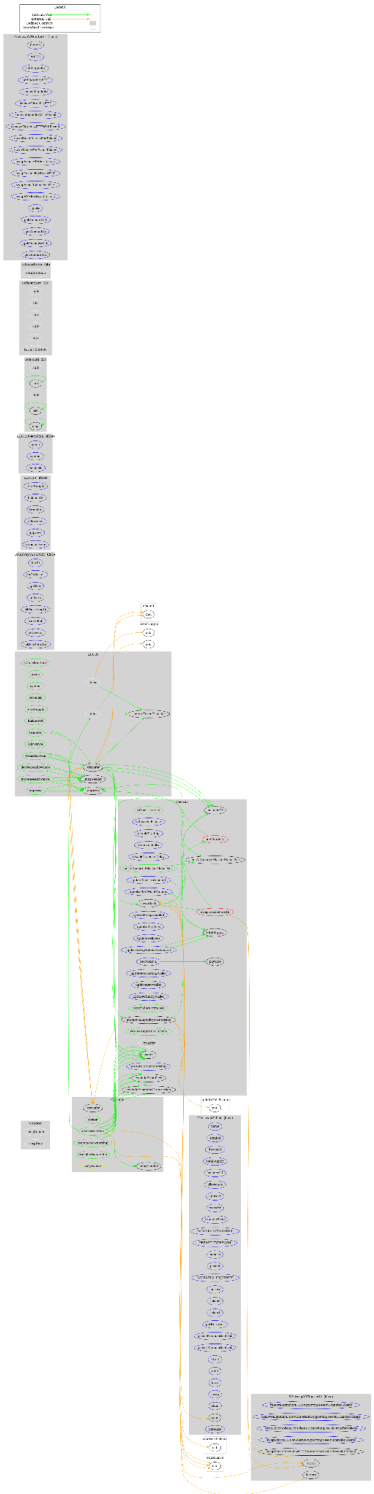
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	updateCharityWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	rescueBNB	External	✓	onlyOwner
	rescueAnyBEP20Tokens	Public	✓	onlyOwner
	_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 draining the contract's tokens and transferring funds to the team's wallet. 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

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

Cyberscope

Audit Report

Clrs NFT

May 2023

Network	BSC Testnet
Address	0x963952a38A25C9A8dD86E09Ff89A0DE22c70b832
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 2
Introduction 3
Findings Breakdown 4
Diagnostics 5
L02 - State Variables could be Declared Constant 6
Description 6
Recommendation 6
L04 - Conformance to Solidity Naming Conventions 7
Description 7
Recommendation 8
L09 - Dead Code Elimination 9
Description 9
Recommendation 9
L13 - Divide before Multiply Operation 11
Description 11
Recommendation 11
L15 - Local Scope Variable Shadowing 12
Description 12
Recommendation 12
L17 - Usage of Solidity Assembly 13
Description 13
Recommendation 13
Functions Analysis 14
Inheritance Graph 22
Flow Graph 23
Summary 24
Disclaimer 25
About Cyberscope 26

Review

Explorer	https://testnet.bscscan.com/address/0x963952a38a25c9a8dd86e09ff89a0de22c70b832

Audit Updates

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

Source Files

Filename	SHA256
NFTPresale.sol	14a69f641a693a0ebbc000c82cbe005451a3bc7144160afd72c04dfff8babf70

Introduction

This is a contract for an NFT presale. The NFTs are ERC721 tokens.

The contract owner can configure

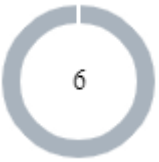
- The minting cost
- The maximum number of NFTs that can be minted per session.
- The NFTs URIs, the base URI extension.
- The whitelisted addresses.
- The mint limit per address.
- The base URIs.
- The base extension of the NFTs URI.

The contract owner has the authority to

- Reveal the not revealed URI.
- Withdraw funds that send 5% of the balance to HashLips address, and the remainder to the contract owner.

Contract users have to pay the mint costs except for the owner who is exempt.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NFTPresale.sol#L1678
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.

```
ess dev = 0x002B2eAD67fE0c3dd4E6eCCB035759848D8D6768;
```

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	NFTPresale.sol#L1652,1702,1726,1766,1770,1774,1779,1783,1787,1791,1795
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.

```
DOMAIN_SEPARATOR
_mintAmount
_owner

_limit
_newCost
_newmaxMintAmount
_newBaseURL
_newBaseExtension
_notRevealedURI
_state
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	NFTPresale.sol#L261,271,290,304,321,331,346,356,371,395,407,442,449,457,468,478,563,581,617,628,670,719,732,762,802,811,826,926,1142,1324,1563,1567,1571,1579,1584,1593,1609
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 functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0,
errorMessage);
    }

...
```

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	NFTPresale.sol#L525,528,540,544,545,546,547,548,549,555
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
prod0 := div(prod0, twos)
result = prod0 * inverse
```

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	NFTPresale.sol#L1687,1688,1726
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.

```
_name,_symbol, _owner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	NFTPresale.sol#L412,489,783,1264
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }


...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC721Metadata	Interface	IERC721		

	name	External		-
	symbol	External		-
	tokenURI	External		-
Address	Library			
	isContract	Internal		
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Math	Library			

	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-

ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	

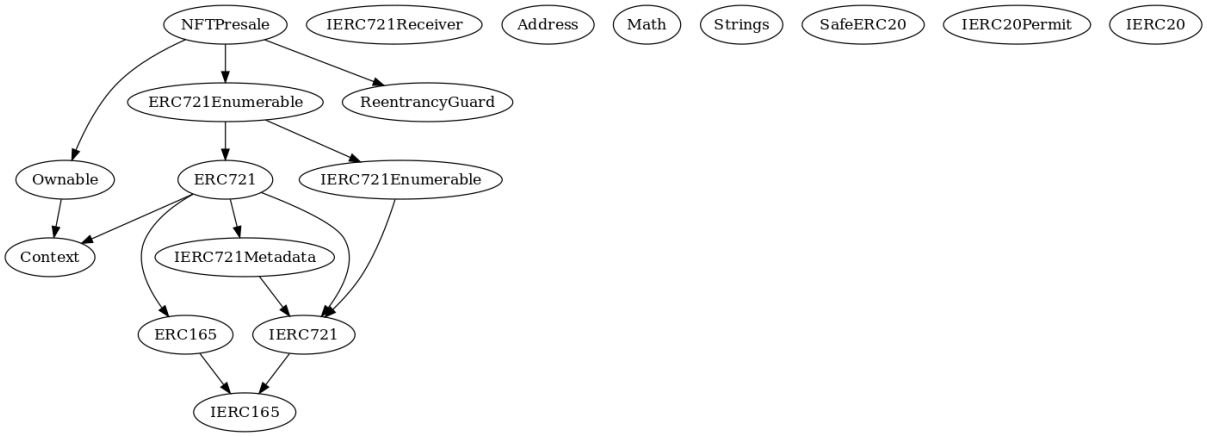
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	_unsafeIncreaseBalance	Internal	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	

	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			

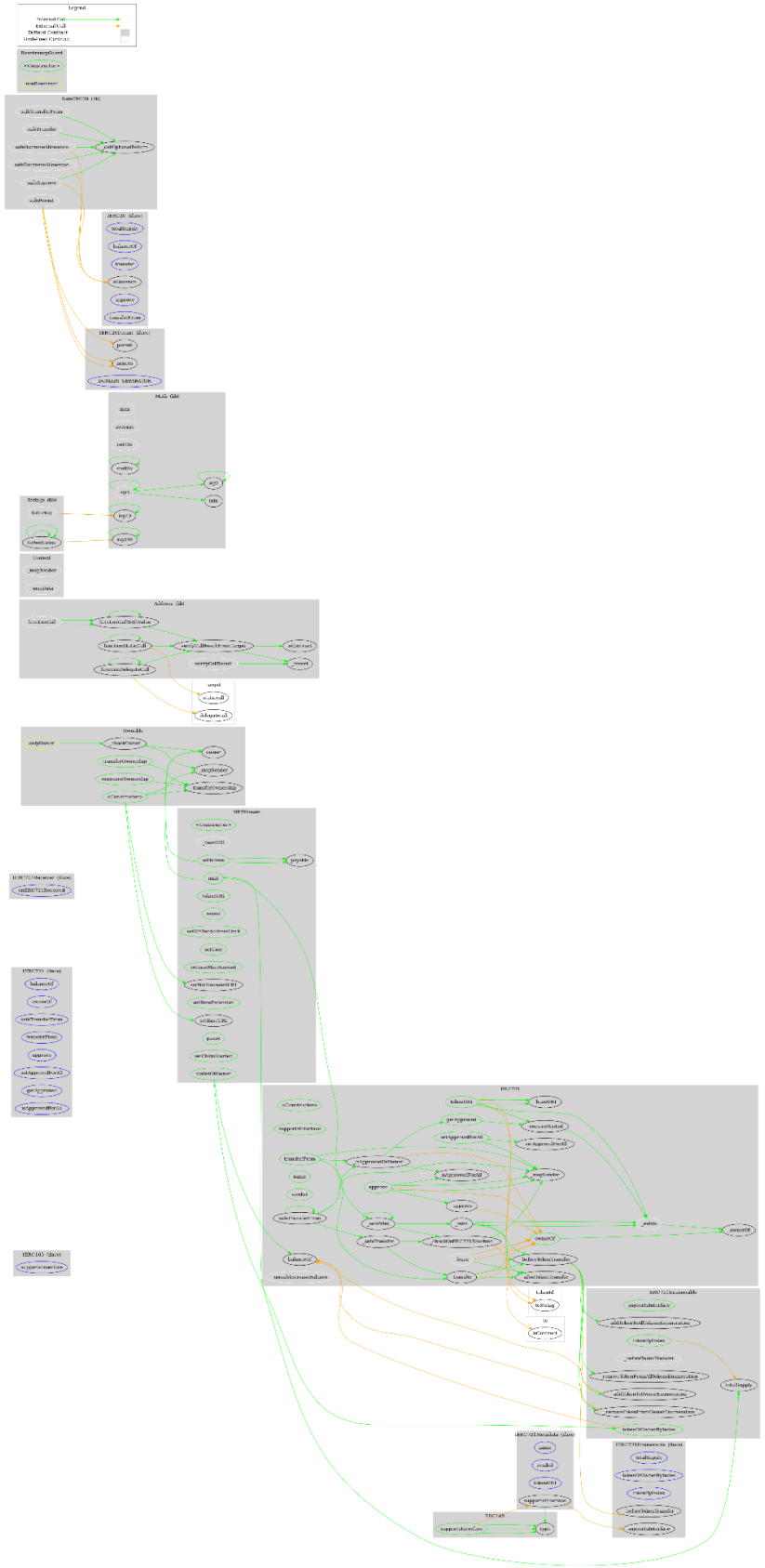
		Public	✓	-
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
NFTPresale	Implementation	ERC721Enumerable, Ownable, ReentrancyGuard		
		Public	✓	ERC721
	_baseURI	Internal		
	mint	Public	Payable	nonReentrant
	walletOfOwner	Public		-
	tokenURI	Public		-
	reveal	Public	✓	onlyOwner

	setNftPerAddressLimit	Public	✓	onlyOwner
	setCost	Public	✓	onlyOwner
	setmaxMintAmount	Public	✓	onlyOwner
	setBaseURI	Public	✓	onlyOwner
	setBaseExtension	Public	✓	onlyOwner
	setNotRevealedURI	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	setClaimStarted	Public	✓	onlyOwner
	withdraw	Public	Payable	onlyOwner nonReentrant

Inheritance Graph



Flow Graph



Summary

ObeseFans Calories contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Clrs Presale

May 2023

Network BSC Testnet

Address 0x2297E5dA9c8922BCD7dfc83872BbE5b7aE91E336

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Introduction 4
Functionality 5
Findings Breakdown 6
Diagnostics 7
MPPC - Missing Purchase Prevalidation Check 8
Description 8
Recommendation 8
PTD - Presale Token Drain 9
Description 9
Recommendation 9
MSE - Missing Solidity Events 10
Description 10
Recommendation 10
IDI - Immutable Declaration Improvement 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 12
L07 - Missing Events Arithmetic 14
Description 14
Recommendation 14
L09 - Dead Code Elimination 15
Description 15
Recommendation 16
L11 - Unnecessary Boolean equality 17
Description 17
Recommendation 17
L16 - Validate Variable Setters 18
Description 18
Recommendation 18
L17 - Usage of Solidity Assembly 19
Description 19
Recommendation 19
Functions Analysis 20

Inheritance Graph **24**
Flow Graph **25**
Summary **26**
Disclaimer **27**
About Cyberscope **28**

Review

Explorer	https://testnet.bscscan.com/address/0x2b675b486a2381384abfb41ee135d06d9093e628

Audit Updates

Initial Audit	18 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v1/presale.pdf
Corrected Phase 2	12 May 2023

Source Files

Filename	SHA256
contracts/CLRSPresale.sol	a70c5e57f04cd95793386cdca7a5ce148b6751d9e7d98983ccefe1a55146b880

Introduction

The CLRSPresale smart contract implements a presale contract for a token.

The contract defines several state variables

- `_contributions`, which is a mapping of addresses to the amount of ether contributed by each address; _token, which is the token being sold.
- `_tokenDecimals`, which is the number of decimals of the token; _wallet, which is the address that will receive the funds raised; _rate, which is the number of tokens that can be purchased for each wei of ether;
- `_weiRaised`, which is the total amount of wei raised.
- `endICO`, which is the end time of the pre-sale event.
- `minPurchase`, which is the minimum amount of ether that can be purchased by a user.
- `maxPurchase`, which is the maximum amount of ether that can be contributed by an address.
- `hardCap`, which is the maximum amount of ether that can be raised.
- `softCap`, which is the minimum amount of ether that needs to be raised.
- `availableTokensICO`, which is the number of tokens available for sale during the pre-sale event.
- `startRefund`, which is a boolean flag indicating whether a refund process has started.
- `refundStartDate`, which is the start time of the refund process.

Functionality

The `startICO` function is used to start the pre-sale event with the necessary arguments.

The `stopICO` function is used to stop the pre-sale event. If the soft cap has been reached, the funds raised are transferred to the wallet address. Otherwise, the `startRefund` flag is set to true, and the `refundStartDate` is set to the current time.

The `buyTokens` function is used to purchase tokens during the pre-sale event. It performs the necessary check to validate that the purchase is valid. Then the number of tokens purchased is calculated based on the `_rate` and `_tokenDecimals` variables, the necessary updates are taking place, and the `TokensPurchased` event is emitted.

The `claimTokens` function is used to claim tokens after the pre-sale event has ended. If the presale is in the correct valid state, the number of tokens purchased by the sender is calculated. The necessary state variables are updated. Finally, the tokens are transferred to the sender, and the `TokensClaimed` event is emitted.

The `withdraw` function is accessible by the owner to withdraw the ether raised during the pre-sale event. If the startRefund flag is false, and either the refund period has ended or no refunds have been requested, the ethers are transferred to the wallet address, and 5% of the ether are transferred to a specific address.

Findings Breakdown



Critical	0
Medium	2
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	2	0	0	0
Minor / Informative	9	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	MPPC	Missing Purchase Prevalidation Check	Unresolved
Medium	PTD	Presale Token Drain	Unresolved
Minor / Informative	MSE	Missing Solidity Events	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L11	Unnecessary Boolean equality	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved

MPPC - Missing Purchase Prevalidation Check

Criticality	Medium
Location	contracts/CLRSPresale.sol#L372
Status	Unresolved

Description

The contract allows users to buy tokens despite the issuance of a refund, which can lead to investors making invalid purchases and potential loss of funds.

```
function _preValidatePurchase(
    address beneficiary,
    uint256 weiAmount
  ) internal view {
    require(
      beneficiary != address(0),
      "Crowdsale: beneficiary is the zero address"
    );
    require(weiAmount != 0, "Crowdsale: weiAmount is 0");
    require(weiAmount >= minPurchase, "have to send at least:
minPurchase");
    require(
      _contributions[beneficiary].add(weiAmount) <=
maxPurchase,
      "can't buy more than: maxPurchase"
    );
    require((_weiRaised + weiAmount) <= hardCap, "Hard Cap
reached");
    this;
}
```

Recommendation

It is recommended to add two extra pre-validation requirements in the token purchase function.

- Checking the `startRefund`, to prevent invalid purchases.

PTD - Presale Token Drain

Criticality	Medium
Location	contracts/CLRSPresale.sol#L452
Status	Unresolved

Description

The contract currently allows the owner to withdraw tokens from the presale contract, which can lead to a reduction in the amount of tokens available for sale during the presale period. This can result in a loss of funds for investors who have already purchased tokens during the presale.

```
function takeTokens(IERC20 tokenAddress)  public onlyOwner
icoNotActive {
    IERC20 tokenBEP = tokenAddress;
    uint256 tokenAmt = tokenBEP.balanceOf(address(this));
    require(tokenAmt > 0, 'BEP-20 balance is 0');
    tokenBEP.safeTransfer(_wallet, tokenAmt);
}
```

Recommendation

It is recommended to remove functionality that allows the contract owner to withdraw tokens from the presale contract. The presale contract should only allow for token purchases by investors and the contract owner should not have the ability to interfere with the presale process.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L400,421,461
Status	Unresolved

Description

The contract has some actions that a user can take on the contract that do not emit any 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.

```
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp);
require(_wallet != address(0));
require(msg.sender != address(0));
```

Recommendation

It is recommend adding events to the code that are emitted whenever a user interacts with the contract in a significant way. These events should include the 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L362,363
Status	Unresolved

Description

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

```
_token
_tokenDecimals
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L310,330,332,334,335,336,379
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);
mapping (address => uint256) public _contributions
IERC20 public _token
address payable public _wallet
uint256 public _rate
uint256 public _weiRaised
uint _hardCap
uint _maxPurchase
uint _minPurchase
uint _softCap
```

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	contracts/CLRSPresale.sol#L388
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.

```
endICO = endDate
```

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/CLRSPresale.sol#L141,148,160,175,179,188,192,217,248,252,260,265,274
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L429,446,502
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(startRefund == false)
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp)
require(startRefund == true, 'no refund available')
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L472,509
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.

```
_wallet = newWallet
recipient.transfer(amount)
```

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/CLRSPresale.sol#L231
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 {
                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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
		Public	✓	-
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	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	

	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
CLRSPresale	Implementation	ReentrancyGuard, Context, Ownable		
		Public	✓	-
		External	Payable	-
	startICO	External	✓	onlyOwner icoNotActive
	stopICO	External	✓	onlyOwner icoActive

	buyTokens	Public	Payable	nonReentrant icoActive
	_preValidatePurchase	Internal		
	claimTokens	External	✓	icoNotActive nonReentrant
	_getTokenAmount	Internal		
	_forwardFunds	Internal	✓	
	withdraw	External	✓	onlyOwner icoNotActive
	checkContribution	Public		-
	setRate	External	✓	onlyOwner icoNotActive
	setAvailableTokens	Public	✓	onlyOwner icoNotActive
	weiRaised	Public		-
	setWalletReceiver	External	✓	onlyOwner
	setHardCap	External	✓	onlyOwner
	setSoftCap	External	✓	onlyOwner
	setMaxPurchase	External	✓	onlyOwner
	setMinPurchase	External	✓	onlyOwner
	takeTokens	Public	✓	onlyOwner icoNotActive
	refundMe	Public	✓	icoNotActive nonReentrant

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
SafeMath (lib)
add
sub
mul
div
mod
Ownable
<Constructor>
owner
onlyOwner
transferOwnership
Context
_msgSender
payable
ReentrancyGuard
<Constructor>
nonReentrant
_wallet
transfer
CLRSPresale
<Constructor>
<Receive Ether>
forwardFunds
stopICO
refundMe
takeTokens
checkContribution
setRate
setAvailableTokens
weiRaised
setWalletReceiver
setHardCap
setSoftCap
setMaxPurchase
setMinPurchase
startICO
withdraw
icoActive
icoNotActive
buyTokens
preValidatePurchase
getTokenAmount
claimTokens
weiAmount
mul
recipient
transfer
_weiRaised
add
IERC20 (iface)
totalSupply
safeTransfer
transfer
balanceOf
approve
transferFrom
allowance
SafeERC20 (lib)
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
callOptionalReturn
safeTransfer
safeTransferFrom
safePermit
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR

Summary

ObeseFans Calories contract implements a presale mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Clrs NFT

May 2023

Network	BSC Testnet
Address	0x963952a38A25C9A8dD86E09Ff89A0DE22c70b832
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 2
Introduction 3
Findings Breakdown 4
Diagnostics 5
L02 - State Variables could be Declared Constant 6
Description 6
Recommendation 6
L04 - Conformance to Solidity Naming Conventions 7
Description 7
Recommendation 8
L09 - Dead Code Elimination 9
Description 9
Recommendation 9
L13 - Divide before Multiply Operation 11
Description 11
Recommendation 11
L15 - Local Scope Variable Shadowing 12
Description 12
Recommendation 12
L17 - Usage of Solidity Assembly 13
Description 13
Recommendation 13
Functions Analysis 14
Inheritance Graph 22
Flow Graph 23
Summary 24
Disclaimer 25
About Cyberscope 26

Review

Explorer	https://testnet.bscscan.com/address/0x963952a38a25c9a8dd86e09ff89a0de22c70b832

Audit Updates

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

Source Files

Filename	SHA256
NFTPresale.sol	14a69f641a693a0ebbc000c82cbe005451a3bc7144160afd72c04dfff8babf70

Introduction

This is a contract for an NFT presale. The NFTs are ERC721 tokens.

The contract owner can configure

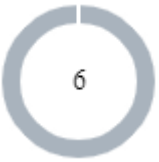
- The minting cost
- The maximum number of NFTs that can be minted per session.
- The NFTs URIs, the base URI extension.
- The whitelisted addresses.
- The mint limit per address.
- The base URIs.
- The base extension of the NFTs URI.

The contract owner has the authority to

- Reveal the not revealed URI.
- Withdraw funds that send 5% of the balance to HashLips address, and the remainder to the contract owner.

Contract users have to pay the mint costs except for the owner who is exempt.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NFTPresale.sol#L1678
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.

```
ess dev = 0x002B2eAD67fE0c3dd4E6eCCB035759848D8D6768;
```

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	NFTPresale.sol#L1652,1702,1726,1766,1770,1774,1779,1783,1787,1791,1795
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.

```
DOMAIN_SEPARATOR
_mintAmount
_owner

_limit
_newCost
_newmaxMintAmount
_newBaseURL
_newBaseExtension
_notRevealedURI
_state
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	NFTPresale.sol#L261,271,290,304,321,331,346,356,371,395,407,442,449,457,468,478,563,581,617,628,670,719,732,762,802,811,826,926,1142,1324,1563,1567,1571,1579,1584,1593,1609
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 functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0,
errorMessage);
    }

...
```

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	NFTPresale.sol#L525,528,540,544,545,546,547,548,549,555
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
prod0 := div(prod0, twos)
result = prod0 * inverse
```

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	NFTPresale.sol#L1687,1688,1726
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.

```
_name,_symbol, _owner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	NFTPresale.sol#L412,489,783,1264
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }


...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC721Metadata	Interface	IERC721		

	name	External		-
	symbol	External		-
	tokenURI	External		-
Address	Library			
	isContract	Internal		
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Math	Library			

	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-

ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	

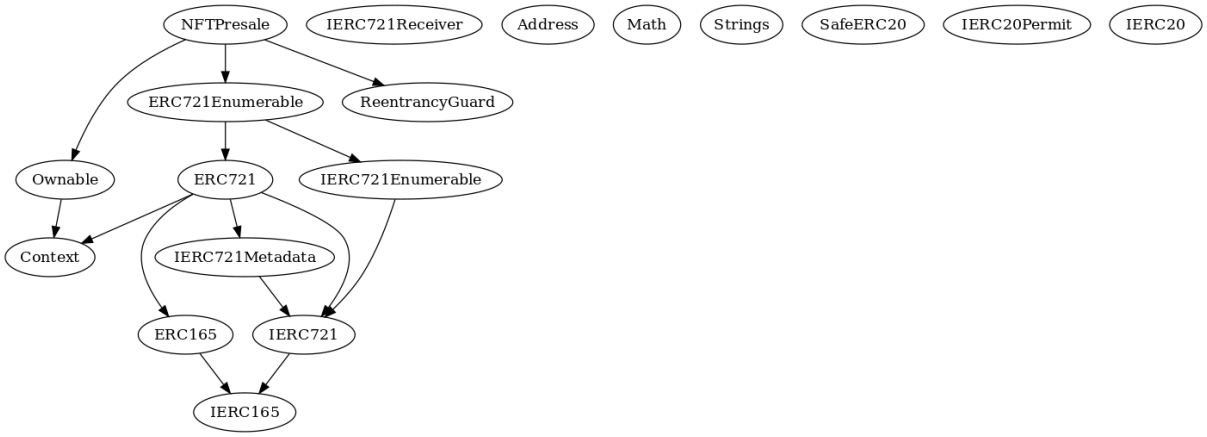
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	_unsafeIncreaseBalance	Internal	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	

	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			

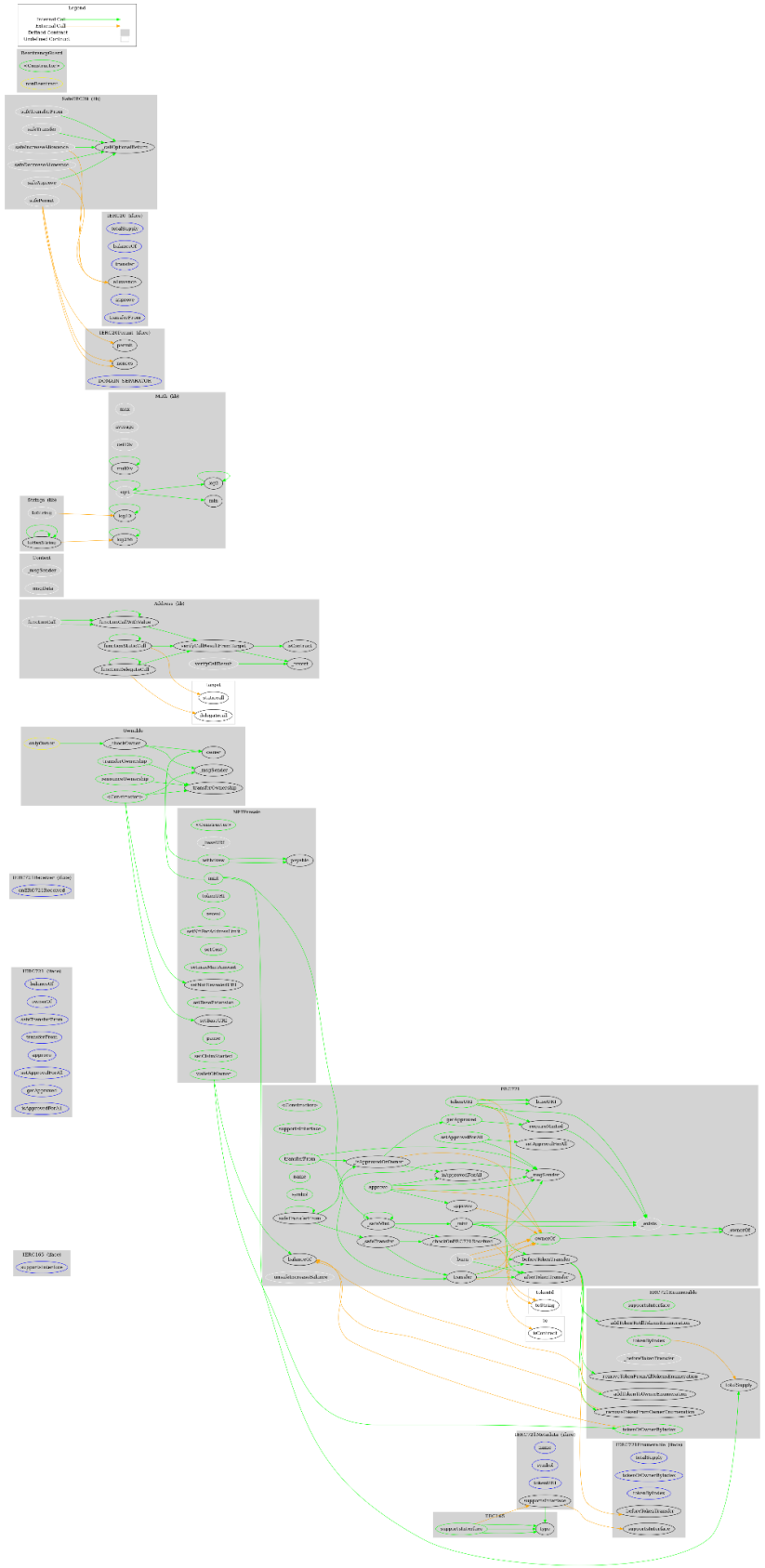
		Public	✓	-
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
NFTPresale	Implementation	ERC721Enumerable, Ownable, ReentrancyGuard		
		Public	✓	ERC721
	_baseURI	Internal		
	mint	Public	Payable	nonReentrant
	walletOfOwner	Public		-
	tokenURI	Public		-
	reveal	Public	✓	onlyOwner

	setNftPerAddressLimit	Public	✓	onlyOwner
	setCost	Public	✓	onlyOwner
	setmaxMintAmount	Public	✓	onlyOwner
	setBaseURI	Public	✓	onlyOwner
	setBaseExtension	Public	✓	onlyOwner
	setNotRevealedURI	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	setClaimStarted	Public	✓	onlyOwner
	withdraw	Public	Payable	onlyOwner nonReentrant

Inheritance Graph



Flow Graph



Summary

ObeseFans Calories contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Clrs Presale

May 2023

Network BSC Testnet

Address 0x2297E5dA9c8922BCD7dfc83872BbE5b7aE91E336

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Introduction 4
Functionality 5
Findings Breakdown 6
Diagnostics 7
MPPC - Missing Purchase Prevalidation Check 8
Description 8
Recommendation 8
PTD - Presale Token Drain 9
Description 9
Recommendation 9
MSE - Missing Solidity Events 10
Description 10
Recommendation 10
IDI - Immutable Declaration Improvement 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 12
L07 - Missing Events Arithmetic 14
Description 14
Recommendation 14
L09 - Dead Code Elimination 15
Description 15
Recommendation 16
L11 - Unnecessary Boolean equality 17
Description 17
Recommendation 17
L16 - Validate Variable Setters 18
Description 18
Recommendation 18
L17 - Usage of Solidity Assembly 19
Description 19
Recommendation 19
Functions Analysis 20

Inheritance Graph **24**
Flow Graph **25**
Summary **26**
Disclaimer **27**
About Cyberscope **28**

Review

Explorer	https://testnet.bscscan.com/address/0x2b675b486a2381384abfb41ee135d06d9093e628

Audit Updates

Initial Audit	18 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/clrs/v1/presale.pdf
Corrected Phase 2	12 May 2023

Source Files

Filename	SHA256
contracts/CLRSPresale.sol	a70c5e57f04cd95793386cdca7a5ce148b6751d9e7d98983ccefe1a55146b880

Introduction

The CLRSPresale smart contract implements a presale contract for a token.

The contract defines several state variables

- `_contributions`, which is a mapping of addresses to the amount of ether contributed by each address; _token, which is the token being sold.
- `_tokenDecimals`, which is the number of decimals of the token; _wallet, which is the address that will receive the funds raised; _rate, which is the number of tokens that can be purchased for each wei of ether;
- `_weiRaised`, which is the total amount of wei raised.
- `endICO`, which is the end time of the pre-sale event.
- `minPurchase`, which is the minimum amount of ether that can be purchased by a user.
- `maxPurchase`, which is the maximum amount of ether that can be contributed by an address.
- `hardCap`, which is the maximum amount of ether that can be raised.
- `softCap`, which is the minimum amount of ether that needs to be raised.
- `availableTokensICO`, which is the number of tokens available for sale during the pre-sale event.
- `startRefund`, which is a boolean flag indicating whether a refund process has started.
- `refundStartDate`, which is the start time of the refund process.

Functionality

The `startICO` function is used to start the pre-sale event with the necessary arguments.

The `stopICO` function is used to stop the pre-sale event. If the soft cap has been reached, the funds raised are transferred to the wallet address. Otherwise, the `startRefund` flag is set to true, and the `refundStartDate` is set to the current time.

The `buyTokens` function is used to purchase tokens during the pre-sale event. It performs the necessary check to validate that the purchase is valid. Then the number of tokens purchased is calculated based on the `_rate` and `_tokenDecimals` variables, the necessary updates are taking place, and the `TokensPurchased` event is emitted.

The `claimTokens` function is used to claim tokens after the pre-sale event has ended. If the presale is in the correct valid state, the number of tokens purchased by the sender is calculated. The necessary state variables are updated. Finally, the tokens are transferred to the sender, and the `TokensClaimed` event is emitted.

The `withdraw` function is accessible by the owner to withdraw the ether raised during the pre-sale event. If the startRefund flag is false, and either the refund period has ended or no refunds have been requested, the ethers are transferred to the wallet address, and 5% of the ether are transferred to a specific address.

Findings Breakdown



Critical	0
Medium	2
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	2	0	0	0
Minor / Informative	9	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	MPPC	Missing Purchase Prevalidation Check	Unresolved
Medium	PTD	Presale Token Drain	Unresolved
Minor / Informative	MSE	Missing Solidity Events	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L11	Unnecessary Boolean equality	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved

MPPC - Missing Purchase Prevalidation Check

Criticality	Medium
Location	contracts/CLRSPresale.sol#L372
Status	Unresolved

Description

The contract allows users to buy tokens despite the issuance of a refund, which can lead to investors making invalid purchases and potential loss of funds.

```
function _preValidatePurchase(
    address beneficiary,
    uint256 weiAmount
  ) internal view {
    require(
      beneficiary != address(0),
      "Crowdsale: beneficiary is the zero address"
    );
    require(weiAmount != 0, "Crowdsale: weiAmount is 0");
    require(weiAmount >= minPurchase, "have to send at least:
minPurchase");
    require(
      _contributions[beneficiary].add(weiAmount) <=
maxPurchase,
      "can't buy more than: maxPurchase"
    );
    require((_weiRaised + weiAmount) <= hardCap, "Hard Cap
reached");
    this;
}
```

Recommendation

It is recommended to add two extra pre-validation requirements in the token purchase function.

- Checking the `startRefund`, to prevent invalid purchases.

PTD - Presale Token Drain

Criticality	Medium
Location	contracts/CLRSPresale.sol#L452
Status	Unresolved

Description

The contract currently allows the owner to withdraw tokens from the presale contract, which can lead to a reduction in the amount of tokens available for sale during the presale period. This can result in a loss of funds for investors who have already purchased tokens during the presale.

```
function takeTokens(IERC20 tokenAddress)  public onlyOwner
icoNotActive {
    IERC20 tokenBEP = tokenAddress;
    uint256 tokenAmt = tokenBEP.balanceOf(address(this));
    require(tokenAmt > 0, 'BEP-20 balance is 0');
    tokenBEP.safeTransfer(_wallet, tokenAmt);
}
```

Recommendation

It is recommended to remove functionality that allows the contract owner to withdraw tokens from the presale contract. The presale contract should only allow for token purchases by investors and the contract owner should not have the ability to interfere with the presale process.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L400,421,461
Status	Unresolved

Description

The contract has some actions that a user can take on the contract that do not emit any 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.

```
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp);
require(_wallet != address(0));
require(msg.sender != address(0));
```

Recommendation

It is recommend adding events to the code that are emitted whenever a user interacts with the contract in a significant way. These events should include the 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L362,363
Status	Unresolved

Description

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

```
_token
_tokenDecimals
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L310,330,332,334,335,336,379
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);
mapping (address => uint256) public _contributions
IERC20 public _token
address payable public _wallet
uint256 public _rate
uint256 public _weiRaised
uint _hardCap
uint _maxPurchase
uint _minPurchase
uint _softCap
```

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	contracts/CLRSPresale.sol#L388
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.

```
endICO = endDate
```

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/CLRSPresale.sol#L141,148,160,175,179,188,192,217,248,252,260,265,274
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L429,446,502
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(startRefund == false)
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp)
require(startRefund == true, 'no refund available')
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L472,509
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.

```
_wallet = newWallet
recipient.transfer(amount)
```

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/CLRSPresale.sol#L231
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 {
                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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
		Public	✓	-
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	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	

	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
CLRSPresale	Implementation	ReentrancyGuard, Context, Ownable		
		Public	✓	-
		External	Payable	-
	startICO	External	✓	onlyOwner icoNotActive
	stopICO	External	✓	onlyOwner icoActive

	buyTokens	Public	Payable	nonReentrant icoActive
	_preValidatePurchase	Internal		
	claimTokens	External	✓	icoNotActive nonReentrant
	_getTokenAmount	Internal		
	_forwardFunds	Internal	✓	
	withdraw	External	✓	onlyOwner icoNotActive
	checkContribution	Public		-
	setRate	External	✓	onlyOwner icoNotActive
	setAvailableTokens	Public	✓	onlyOwner icoNotActive
	weiRaised	Public		-
	setWalletReceiver	External	✓	onlyOwner
	setHardCap	External	✓	onlyOwner
	setSoftCap	External	✓	onlyOwner
	setMaxPurchase	External	✓	onlyOwner
	setMinPurchase	External	✓	onlyOwner
	takeTokens	Public	✓	onlyOwner icoNotActive
	refundMe	Public	✓	icoNotActive nonReentrant

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
SafeMath (lib)
add
sub
mul
div
mod
Ownable
<Constructor>
owner
onlyOwner
transferOwnership
Context
_msgSender
payable
ReentrancyGuard
<Constructor>
nonReentrant
_wallet
transfer
CLRSPresale
<Constructor>
<Receive Ether>
forwardFunds
stopICO
refundMe
takeTokens
checkContribution
setRate
setAvailableTokens
weiRaised
setWalletReceiver
setHardCap
setSoftCap
setMaxPurchase
setMinPurchase
startICO
withdraw
icoActive
icoNotActive
buyTokens
preValidatePurchase
getTokenAmount
claimTokens
weiAmount
mul
recipient
transfer
_weiRaised
add
IERC20 (iface)
totalSupply
safeTransfer
transfer
balanceOf
approve
transferFrom
allowance
SafeERC20 (lib)
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
callOptionalReturn
safeTransfer
safeTransferFrom
safePermit
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR

Summary

ObeseFans Calories contract implements a presale mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

ObeseFans Calories

April 2023

SHA256 2f3a940b2aeeb3d828af9e62604f65f7eb44444ac4acacb8612a0d7f1243d836

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Findings Breakdown **4**
Analysis **5**
OCTD - Transfers Contract's Tokens 6
Description 6
Recommendation 6
ULTW - Transfers Liquidity to Team Wallet 7
Description 7
Recommendation 7
BT - Burns Tokens 8
Description 8
Recommendation 9
Diagnostics **10**
AFI - Accumulated Fees Inconsistency 11
Description 11
Recommendation 11
RSML - Redundant SafeMath Library 12
Description 12
Recommendation 12
L02 - State Variables could be Declared Constant 13
Description 13
Recommendation 13
L04 - Conformance to Solidity Naming Conventions 14
Description 14
Recommendation 15
L05 - Unused State Variable 16
Description 16
Recommendation 16
L07 - Missing Events Arithmetic 17
Description 17
Recommendation 17
L08 - Tautology or Contradiction 18
Description 18
Recommendation 18
L09 - Dead Code Elimination 19
Description 19

Recommendation 19
L13 - Divide before Multiply Operation 21
Description 21
Recommendation 21
L15 - Local Scope Variable Shadowing 22
Description 22
Recommendation 22
L16 - Validate Variable Setters 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Contract Name	Calories
Testing Deploy	https://testnet.bscscan.com/address/0x4ca6812c88db2c319c20bfabc0cfa53f7cd5921f
Symbol	$CLRS
Decimals	18
Total Supply	10.000.000.000

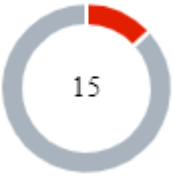
Audit Updates

Initial Audit	11 Apr 2023

Source Files

Filename	SHA256
contracts/CLRS.sol	2f3a940b2aeeb3d828af9e62604f65f7eb44444ac4acacb8612a0d7f1243d836

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Analysis

Critical Medium Minor / Informative Pass

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

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1123
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) public onlyOwner {
  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
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.

BT - Burns Tokens

Criticality	Critical
Location	contracts/CLRS.sol#L1330
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.mul(percentForLPBurn).div(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
●	AFI	Accumulated Fees Inconsistency	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
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

AFI - Accumulated Fees Inconsistency

Criticality	Critical
Location	contracts/CLRS.sol#L1279
Status	Unresolved

Description

The contract has a `rescueAnyBEP20Tokens` function that withdraws tokens from the contract. However, the contract does not update the variable that tracks the accumulated fees when the withdrawal occurs. This can result in incorrect fee calculations and cause the swap function to revert.

```
function swapBack() private {
    uint256 contractBalance = balanceOf(address(this));
    uint256 totalTokensToSwap = tokensForLiquidity + tokensForMarketing
+ tokensForDev + tokensForCharity;
    ...
}
```

Recommendation

It is recommended to update the variables that tracks the accumulated fees every time a token withdrawal occurs.

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/CLRS.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/CLRS.sol#L893,894
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.

```
uint256 public manualBurnFrequency = 30 minutes
uint256 public lastManualLpBurnTime
```

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/CLRS.sol#L47,48,65,738,876,925,937,939,941,1063,1072,1123,1322
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);
address public constant deadAddress = address(0xdead)
mapping (address => bool) public
_isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event devWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event charityWalletUpdated(address indexed newWallet, address
indexed oldWallet);
uint256 _liquidityFee
uint256 _devFee
uint256 _buyCharityFee
uint256 _marketingFee
uint256 _sellCharityFee

...
```

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/CLRS.sol#L669
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	contracts/CLRS.sol#L1040,1046,1051,1064,1073,1325
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
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	contracts/CLRS.sol#L1324
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	contracts/CLRS.sol#L414,715,721,728
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	contracts/CLRS.sol#L1215,1216,1217,1218,1219,1223,1224,1225,1226,1227
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.

```
tokensForMarketing += fees * sellMarketingFee / sellTotalFees
fees = amount.mul(buyTotalFees).div(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	contracts/CLRS.sol#L974
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1100,1105
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.

```
marketingWallet = newMarketingWallet
devWallet = 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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/CLRS.sol#L1124
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		

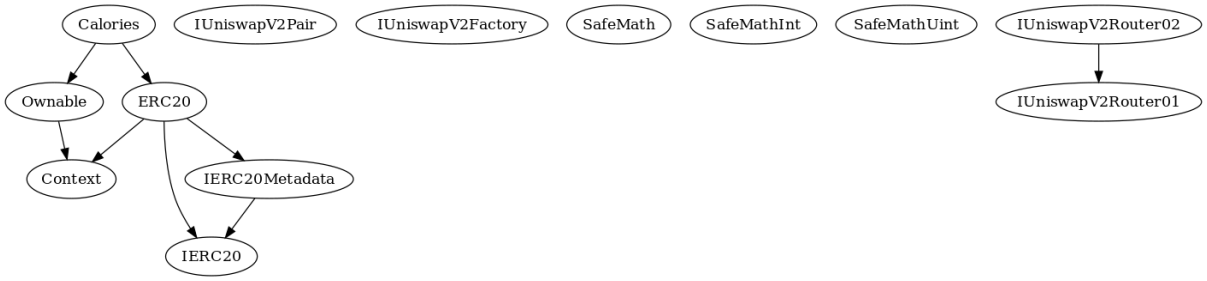
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	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		

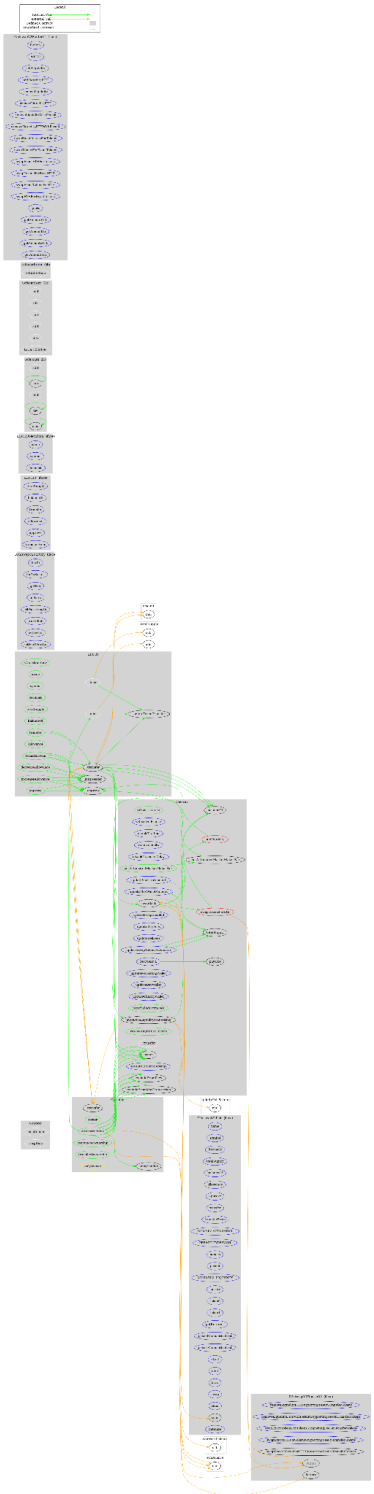
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	updateCharityWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	rescueBNB	External	✓	onlyOwner
	rescueAnyBEP20Tokens	Public	✓	onlyOwner
	_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 draining the contract's tokens and transferring funds to the team's wallet. 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

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

Cyberscope

Audit Report

Clrs NFT

April 2023

SHA256 14a69f641a693a0ebbc000c82cbe005451a3bc7144160afd72c04dfff8babf70

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Findings Breakdown **5**
Diagnostics **6**
FSO - For Statement Optimization 7
Description 7
Recommendation 7
MSC - Missing Sanity Check 8
Description 8
Recommendation 8
DSM - Data Structure Misuse 9
Description 9
Recommendation 9
RSML - Redundant SafeMath Library 10
Description 10
Recommendation 10
L02 - State Variables could be Declared Constant 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 12
L09 - Dead Code Elimination 14
Description 14
Recommendation 14
L11 - Unnecessary Boolean equality 16
Description 16
Recommendation 16
L14 - Uninitialized Variables in Local Scope 17
Description 17
Recommendation 17
L15 - Local Scope Variable Shadowing 18
Description 18
Recommendation 18
L17 - Usage of Solidity Assembly 19
Description 19

Recommendation 19
Functions Analysis **20**
Inheritance Graph **29**
Flow Graph **30**
Summary **31**
Disclaimer **32**
About Cyberscope **33**

Review

Contract Name	NFTPresale
Testing Deploy	https://testnet.bscscan.com/address/0x488b8c3a6c19df49bbf6807aefab1a255251e5f5
Symbol	TST

Audit Updates

Initial Audit	18 Apr 2023

Source Files

Filename	SHA256
contracts/NFTPresale.sol	14a69f641a693a0ebbc000c82cbe005451a3bc7144160afd72c04dfff8babf70

Introduction

This is a contract for an NFT presale. The NFTs are ERC721 tokens.

The contract owner can configure

- The minting cost
- The maximum number of NFTs that can be minted per session.
- The NFTs URIs, the base URI extension.
- The whitelisted addresses.
- The mint limit per address.
- The base URIs.
- The base extension of the NFTs URI.

The contract owner has the authority to

- Reveal the not revealed URI.
- Withdraw funds that send 5% of the balance to HashLips address, and the remainder to the contract owner.

Contract users have to pay the mint costs except for the owner who is exempt.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	FSO	For Statement Optimization	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	DSM	Data Structure Misuse	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

FSO - For Statement Optimization

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1774
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 contract is aggregating the user's minted balance inside a for loop that does not depend on the loop index since there is the variable `_mintAmount`. This means that the statement is being unnecessarily executed multiple times.

```
for (uint256 i = 1; i <= _mintAmount; i++) {
    addressMintedBalance[msg.sender]++;
    _safeMint(msg.sender, supply + i);
}
```

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.

The contract could move the user's minted balance aggregation after the for-loop.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1865,1837
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The whitelisted addresses can be set to zero addresses.

```
function whitelistUsers(address[] calldata _users) public
onlyOwner {
    delete whitelistedAddresses;
    whitelistedAddresses = _users;
}
```

The max mint amount should be greater than 0 since the min mint amount is greater than zero. `require(_mintAmount > 0, "need to mint at least 1 NFT");`

```
function setmaxMintAmount(uint256 _newmaxMintAmount) public
onlyOwner {
    maxMintAmount = _newmaxMintAmount;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications. The argument address should not be set to zero addresses and the ma mint amount should be greater than 0.

DSM - Data Structure Misuse

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1780
Status	Unresolved

Description

The contract uses the valuable `whitelistedAddresses` as an array. The business logic of the contract does not require to iterate this structure sequentially. Thus, unnecessary loops are produced that increase the required gas.

```
function isWhitelisted(address _user) public view returns
(bool) {
    for (uint i = 0; i < whitelistedAddresses.length; i++) {
      if (whitelistedAddresses[i] == _user) {
          return true;
      }
    }
    return false;
}
```

Recommendation

The contract could use a data structure that provides instant access. For instance, a Set or a Map would fit better to the business logic of the contract. This way the time complexity will be reduced from o(n) to o(1).

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/NFTPresale.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/NFTPresale.sol#L1732
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.

```
public maxSupply = 2500;
```

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/NFTPresale.sol#L1381,1709,1758,1780,1789,1829,1833,1837,1841,1845,1849,1853,1857,1861,1865
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.

```
_mintAmount
_user

_owner


...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L293,318,328,347,361,378,388,403,413,428,452,464,499,506,514,525,535,620,638,674,685,727,776,789,819,859,868,883,983,1199,1381,1620,1624,1628,1636,1641,1650,1666
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1766,1814
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
if(Whitelisted == true) {

if(revealed == false) {
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1796
Status	Unresolved

Description

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

```
uint256 i
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L1743,1744,1789
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.

```
string memory _name,

string memory _symbol,

address _owner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/NFTPresale.sol#L469,546,840,1321
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }


...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-

	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-
	tokenURI	External		-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		

	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		

	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-

	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	__unsafe_increaseBalance	Internal	✓	

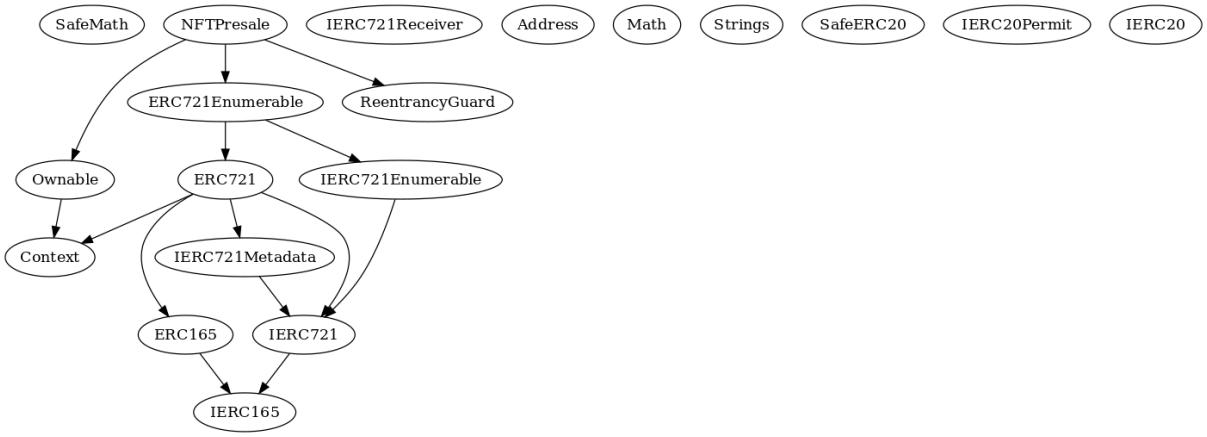
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	
	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

	_transferOwnership	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
ReentrancyGuard	Implementation			
		Public	✓	-
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

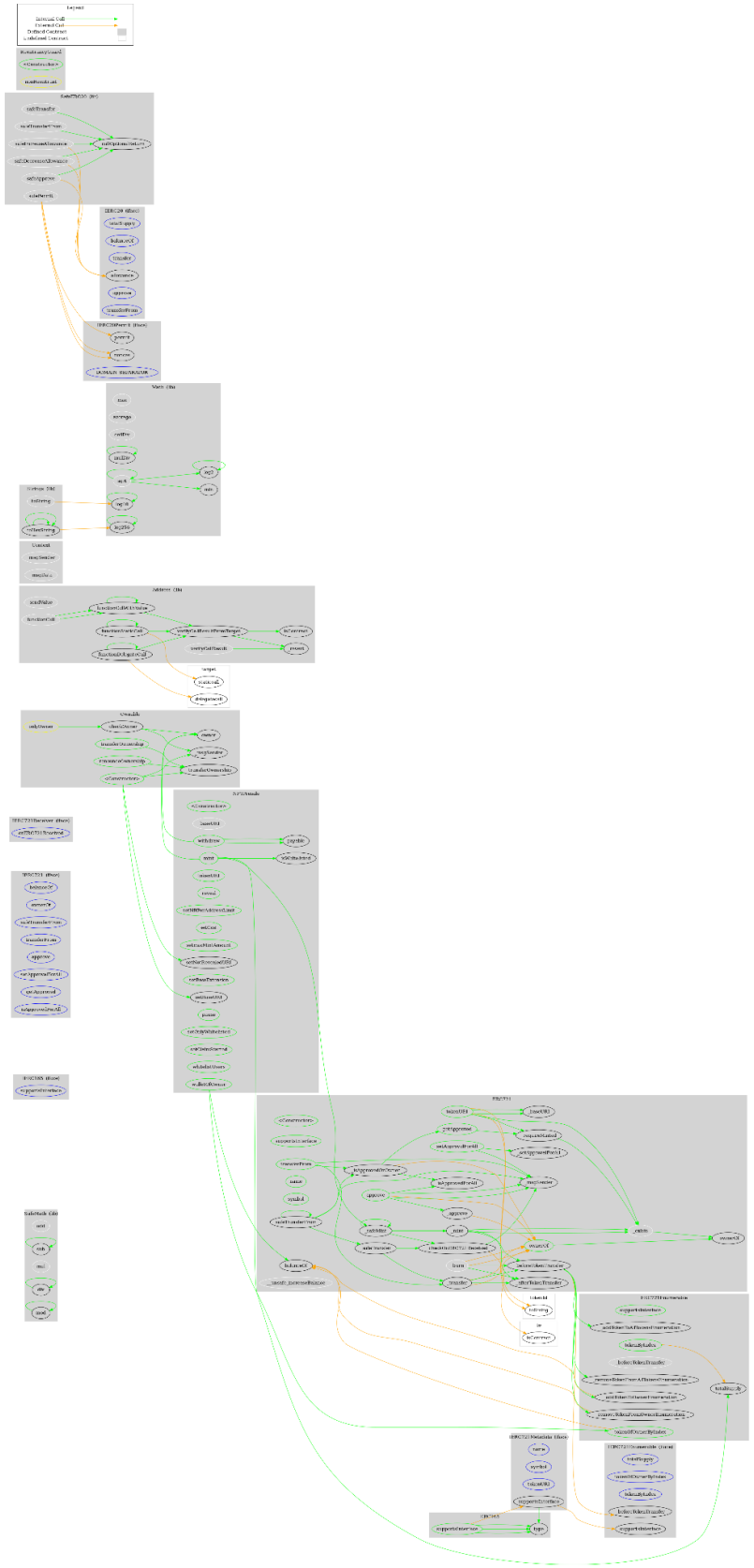
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
NFTPresale	Implementation	ERC721Enumerable, Ownable, ReentrancyGuard		
		Public	✓	ERC721
	_baseURI	Internal		
	mint	Public	Payable	nonReentrant
	isWhitelisted	Public		-
	walletOfOwner	Public		-
	tokenURI	Public		-
	reveal	Public	✓	onlyOwner
	setNftPerAddressLimit	Public	✓	onlyOwner
	setCost	Public	✓	onlyOwner
	setmaxMintAmount	Public	✓	onlyOwner
	setBaseURI	Public	✓	onlyOwner
	setBaseExtension	Public	✓	onlyOwner
	setNotRevealedURI	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	setOnlyWhitelisted	Public	✓	onlyOwner
	setClaimStarted	Public	✓	onlyOwner
	whitelistUsers	Public	✓	onlyOwner

	withdraw	Public	Payable	onlyOwner nonReentrant

Inheritance Graph



Flow Graph



Summary

ObeseFans Calories contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Clrs Presale

April 2023

SHA256 a70c5e57f04cd95793386cdca7a5ce148b6751d9e7d98983ccefe1a55146b880

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Introduction **4**
Functionality 4
Findings Breakdown **6**
Diagnostics **7**
PMI - Presale Mechanism Inconsistency 9
Description 9
Recommendation 9
MSC - Missing Sanity Check 10
Description 10
Recommendation 11
PTD - Presale Token Drain 12
Description 12
Recommendation 12
MPPC - Missing Purchase Prevalidation Check 13
Description 13
Recommendation 14
VO - Variable Optimization 15
Description 15
Recommendation 15
CI - Calculation Inconsistency 16
Description 16
Recommendation 16
MRPC - Missing Refund Prevalidation Check 17
Description 17
Recommendation 17
MSE - Missing Solidity Events 18
Description 18
Recommendation 18
RSML - Redundant SafeMath Library 19
Description 19
Recommendation 19
IDI - Immutable Declaration Improvement 20
Description 20
Recommendation 20
L04 - Conformance to Solidity Naming Conventions 21

Description 21
Recommendation 21
L07 - Missing Events Arithmetic 23
Description 23
Recommendation 23
L09 - Dead Code Elimination 24
Description 24
Recommendation 25
L11 - Unnecessary Boolean equality 26
Description 26
Recommendation 26
L16 - Validate Variable Setters 27
Description 27
Recommendation 27
L17 - Usage of Solidity Assembly 28
Description 28
Recommendation 28
Functions Analysis **29**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Testing Deploy	https://testnet.bscscan.com/address/0x2b675b486a2381384abfb41ee135d06d9093e628

Audit Updates

Initial Audit	18 Apr 2023

Source Files

Filename	SHA256
contracts/CLRSPresale.sol	a70c5e57f04cd95793386cdca7a5ce148b6751d9e7d98983ccefe1a55146b880

Introduction

The CLRSPresale smart contract implements a presale contract for a token.

The contract defines several state variables

- `_contributions`, which is a mapping of addresses to the amount of ether contributed by each address; _token, which is the token being sold.
- `_tokenDecimals`, which is the number of decimals of the token; _wallet, which is the address that will receive the funds raised; _rate, which is the number of tokens that can be purchased for each wei of ether;
- `_weiRaised`, which is the total amount of wei raised.
- `endICO`, which is the end time of the pre-sale event.
- `minPurchase`, which is the minimum amount of ether that can be purchased by a user.
- `maxPurchase`, which is the maximum amount of ether that can be contributed by an address.
- `hardCap`, which is the maximum amount of ether that can be raised.
- `softCap`, which is the minimum amount of ether that needs to be raised.
- `availableTokensICO`, which is the number of tokens available for sale during the pre-sale event.
- `startRefund`, which is a boolean flag indicating whether a refund process has started.
- `refundStartDate`, which is the start time of the refund process.

Functionality

The `startICO` function is used to start the pre-sale event with the necessary arguments.

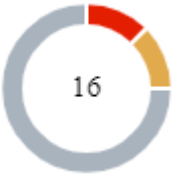
The `stopICO` function is used to stop the pre-sale event. If the soft cap has been reached, the funds raised are transferred to the wallet address. Otherwise, the `startRefund` flag is set to true, and the `refundStartDate` is set to the current time.

The `buyTokens` function is used to purchase tokens during the pre-sale event. It performs the necessary check to validate that the purchase is valid. Then the number of tokens purchased is calculated based on the `_rate` and `_tokenDecimals` variables, the necessary updates are taking place, and the `TokensPurchased` event is emitted.

The `claimTokens` function is used to claim tokens after the pre-sale event has ended. If the presale is in the correct valid state, the number of tokens purchased by the sender is calculated. The necessary state variables are updated. Finally, the tokens are transferred to the sender, and the `TokensClaimed` event is emitted.

The `withdraw` function is accessible by the owner to withdraw the ether raised during the pre-sale event. If the startRefund flag is false, and either the refund period has ended or no refunds have been requested, the ethers are transferred to the wallet address, and 5% of the ether are transferred to a specific address.

Findings Breakdown



Critical	2
Medium	2
Minor / Informative	12

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PMI	Presale Mechanism Inconsistency	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	PTD	Presale Token Drain	Unresolved
●	MPPC	Missing Purchase Prevalidation Check	Unresolved
●	VO	Variable Optimization	Unresolved
●	CI	Calculation Inconsistency	Unresolved
●	MRPC	Missing Refund Prevalidation Check	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

PMI - Presale Mechanism Inconsistency

Criticality	Critical
Location	contracts/CLRSPresale.sol#L461
Status	Unresolved

Description

The presale mechanism in the contract should have consistent parameters that are fixed and cannot be modified by the owner once the presale has started. All parameters, including the available token amount, should be set before the start of the presale to ensure consistency and fairness for all investors.

```
function setAvailableTokens(uint256 amount) public onlyOwner
icoNotActive{
    availableTokensICO = amount;
    emit AvailableTokenUpdated(amount);
}
```

Recommendation

It is recommended to have fixed `availableTokensICO` presale parameter before the start of the presale and prevent the owner from modifying it once the presale has started.

MSC - Missing Sanity Check

Criticality	Critical
Location	contracts/CLRSPresale.sol#L456,461,475,480,484
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The `_rate` variable could be set to zero and prevent users from taking tokens on the purchase transaction.

```
function setRate(uint256 newRate) external onlyOwner icoNotActive {
    _rate = newRate;
    emit RateUpdated(newRate);
}
```

The min purchase value can be set to values greater than the max purchase value and to zero.

```
function setMinPurchase(uint256 value) external onlyOwner{
    minPurchase = value;
    emit MinPurchaseUpdated(value);
}
```

The max purchase amount could be set to values lower than the min purchase amount.

```
function setMaxPurchase(uint256 value) external onlyOwner{
    maxPurchase = value;
    emit MaxPurchaseUpdated(value);
}
```

The hard cap could be set to values lower than the soft cap.

```
function setHardCap(uint256 value) external onlyOwner{
    hardCap = value;
    emit HardCapUpdated(value);
}
```

The soft value could be set to values greater than the hard cap.

```
function setSoftCap(uint256 value) external onlyOwner {
    softCap = value;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The `softCap` variable should exceed the hard cap.
- The `rate` variable should be greater than zero.
- The `maxPurchase` variable should be lower than or equal to `hardCap` .

The following requirements are used in the contract's implementation. The contract should re-apply the following requirements to the related setter functions.

- `require(_softCap < _hardCap, "Softcap must be lower than Hardcap");`
- `require(_minPurchase < _maxPurchase, "minPurchase must be lower than maxPurchase");`
- `require(_minPurchase > 0, '_minPurchase should > 0');`

PTD - Presale Token Drain

Criticality	Medium
Location	contracts/CLRSPresale.sol#L494
Status	Unresolved

Description

The contract currently allows the owner to withdraw tokens from the presale contract, which can lead to a reduction in the amount of tokens available for sale during the presale period. This can result in a loss of funds for investors who have already purchased tokens during the presale.

```
function takeTokens(IERC20 tokenAddress)  public onlyOwner
icoNotActive {
    IERC20 tokenBEP = tokenAddress;
    uint256 tokenAmt = tokenBEP.balanceOf(address(this));
    require(tokenAmt > 0, 'BEP-20 balance is 0');
    tokenBEP.safeTransfer(_wallet, tokenAmt);
}
```

Recommendation

It is recommended to remove functionality that allows the contract owner to withdraw tokens from the presale contract. The presale contract should only allow for token purchases by investors and the contract owner should not have the ability to interfere with the presale process.

MPPC - Missing Purchase Prevalidation Check

Criticality	Medium
Location	contracts/CLRSPresale.sol#L419
Status	Unresolved

Description

The contract allows users to buy tokens despite the issuance of a refund, which can lead to investors making invalid purchases and potential loss of funds.

```
function _preValidatePurchase(
    address beneficiary,
    uint256 weiAmount
  ) internal view {
    require(
      beneficiary != address(0),
      "Crowdsale: beneficiary is the zero address"
    );
    require(weiAmount != 0, "Crowdsale: weiAmount is 0");
    require(weiAmount >= minPurchase, "have to send at least:
minPurchase");
    require(
      _contributions[beneficiary].add(weiAmount) <=
maxPurchase,
      "can't buy more than: maxPurchase"
    );
    require((_weiRaised + weiAmount) <= hardCap, "Hard Cap
reached");
    this;
}
```

The contract allows the users to buy tokens even if there are not enough tokens for purchase.

```
function buyTokens(address beneficiary) public nonReentrant
icoActive payable {
  ...
  availableTokensICO = availableTokensICO - tokens;
  ...
}
```

Recommendation

It is recommended to add two extra pre-validation requirements in the token purchase function.

- Checking the `startRefund`, to prevent invalid purchases.
- Checking if there are enough available tokens for purchase.

VO - Variable Optimization

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L363
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 variable `tokenDecimals` is represented in `uint256`. The corresponding number will never exceed the value of 18, then the implementation of the contract does not necessarily require the use of a uint256 data type.

```
constructor (uint256 rate, address payable wallet, IERC20 token,
uint256 tokenDecimals)  {
    ...
    _tokenDecimals = 18 - tokenDecimals;
}
```

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.

The variable `tokenDecimals` could be represented to `uint8`.

CI - Calculation Inconsistency

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L413,414
Status	Unresolved

Description

The contract performs calculations using two different methods, either native calculations or the SafeMath library, which can make it difficult to read and understand. This inconsistency may also introduce errors or vulnerabilities in the contract's logic.

```
_weiRaised = _weiRaised.add(weiAmount);
availableTokensICO = availableTokensICO - tokens;
```

Recommendation

It is recommended to use native calculations since the contract is compiled using a version of Solidity greater than 0.8.0. The compiler performs calculation checks to prevent overflows and underflows, making it safe to use native calculations. This will make the code simpler and easier to read and understand.

MRPC - Missing Refund Prevalidation Check

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L501
Status	Unresolved

Description

The contract attempts to refund a user without verifying if the contract has enough balance to refund the user or if the user has any other contribution to refund.

```
function refundMe() public icoNotActive nonReentrant {
    require(startRefund == true, 'no refund available');
    require(msg.sender != address(0));
    uint amount = _contributions[msg.sender];
    if (address(this).balance >= amount) {
        _contributions[msg.sender] = 0;
        if (amount > 0) {
            address payable recipient = payable(msg.sender);
            recipient.transfer(amount);
            emit Refund(msg.sender, amount);
        }
    }
}
```

Recommendation

It is recommended to add pre-validation checks with require statements when attempting to refund a user. The checks should verify that the contract has enough balance to refund the user and that the user has contributions to refund. If either of these conditions are not met, the contract should revert with a clear error message indicating the reason for the revert.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L429,446,471,503
Status	Unresolved

Description

The contract has some actions that a user can take on the contract that do not emit any 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.

```
require(startRefund == false);
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp);
require(_wallet != address(0));
require(msg.sender != address(0));
```

Recommendation

It is recommended adding events to the code that are emitted whenever a user interacts with the contract in a significant way. 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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/CLRSPresale.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L362,363
Status	Unresolved

Description

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

```
_token
_tokenDecimals
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L310,330,332,334,335,336,379
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);
mapping (address => uint256) public _contributions
IERC20 public _token
address payable public _wallet
uint256 public _rate
uint256 public _weiRaised
uint _hardCap
uint _maxPurchase
uint _minPurchase
uint _softCap
```

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	contracts/CLRSPresale.sol#L388
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.

```
endICO = endDate
```

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/CLRSPresale.sol#L141,148,160,175,179,188,192,217,248,252,260,265,274
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L429,446,502
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(startRefund == false)
require(startRefund == false || (refundStartDate + 3 days) <
block.timestamp)
require(startRefund == true, 'no refund available')
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/CLRSPresale.sol#L472,509
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.

```
_wallet = newWallet
recipient.transfer(amount)
```

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/CLRSPresale.sol#L231
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 {
                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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
		Public	✓	-
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	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	

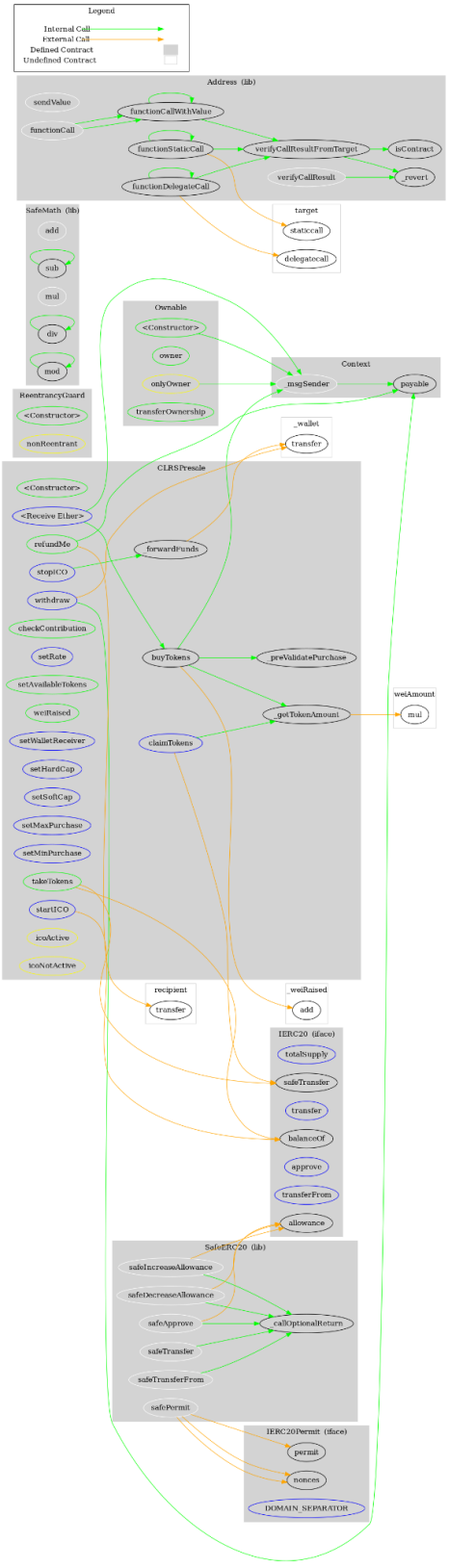
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
CLRSPresale	Implementation	ReentrancyGuard, Context, Ownable		
		Public	✓	-
		External	Payable	-
	startICO	External	✓	onlyOwner icoNotActive
	stopICO	External	✓	onlyOwner icoActive

	buyTokens	Public	Payable	nonReentrant icoActive
	_preValidatePurchase	Internal		
	claimTokens	External	✓	icoNotActive nonReentrant
	_getTokenAmount	Internal		
	_forwardFunds	Internal	✓	
	withdraw	External	✓	onlyOwner icoNotActive
	checkContribution	Public		-
	setRate	External	✓	onlyOwner icoNotActive
	setAvailableTokens	Public	✓	onlyOwner icoNotActive
	weiRaised	Public		-
	setWalletReceiver	External	✓	onlyOwner
	setHardCap	External	✓	onlyOwner
	setSoftCap	External	✓	onlyOwner
	setMaxPurchase	External	✓	onlyOwner
	setMinPurchase	External	✓	onlyOwner
	takeTokens	Public	✓	onlyOwner icoNotActive
	refundMe	Public	✓	icoNotActive nonReentrant

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
_revert
target
staticcall
delegatecall
SafeMath (lib)
add
sub
mul
div
mod
Ownable
<Constructor>
owner
onlyOwner
transferOwnership
Context
msgSender
payable
ReentrancyGuard
<Constructor>
nonReentrant
_wallet
transfer
CLRSPresale
<Constructor>
<Receive Ether>
refundMe
stopICO
withdraw
checkContribution
setRate
setAvailableTokens
weiRaised
setWalletReceiver
setHardCap
setSoftCap
setMaxPurchase
setMinPurchase
takeTokens
startICO
icoActive
icoNotActive
forwardFunds
buyTokens
preValidatePurchase
_getTokenAmount
claimTokens
weiAmount
mul
recipient
transfer
weiRaised
add
IERC20 (iface)
totalSupply
safeTransfer
transfer
balanceOf
approve
transferFrom
allowance
SafeERC20 (lib)
safeIncreaseAllowance
safeDecreaseAllowance
safeApprove
safeTransfer
safeTransferFrom
safePermit
callOptionalReturn
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR

Summary

ObeseFans Calories contract implements a presale mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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