



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.

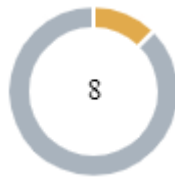
The `stopICO` function is used to stop the pre-sale event. If the soft cap has been reached, the funds raised are transferred to the wallet address. Otherwise, the `startRefund` flag is set to true, and the `refundStartDate` is set to the current time.

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

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	✓	

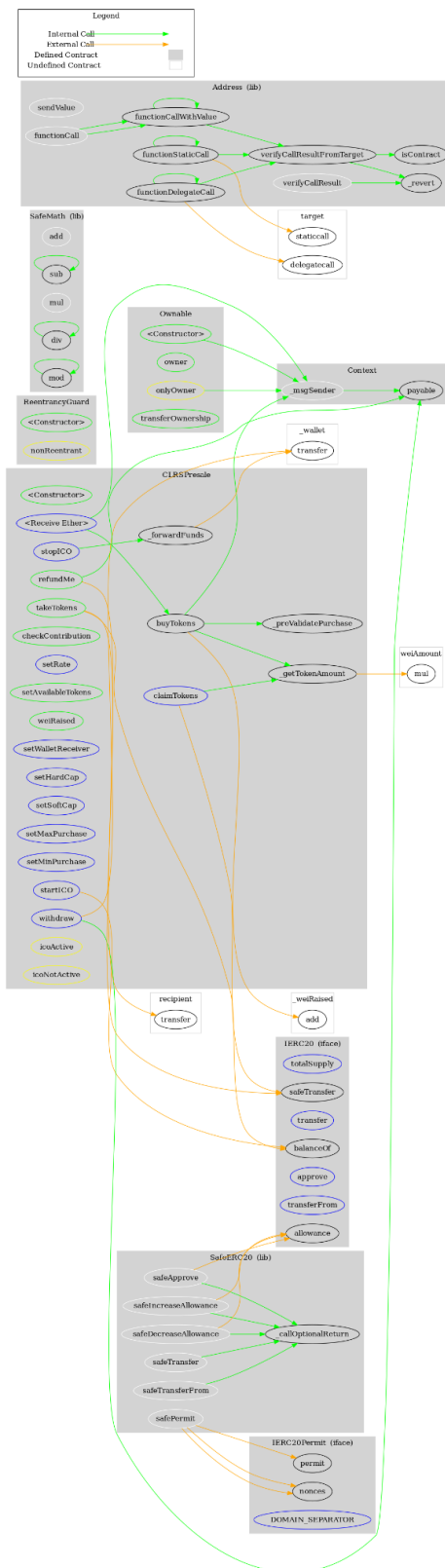
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	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



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>