Cyberscope

Audit Report

INTDESTCOIN

May 2023

SHA256 35e1b7a992bc0de86ce4f8c7da7f132450ddb7ee4454be9fa5e3598a91c79927

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0xc8c875abaa8379cb28f62859e99603e3ca7bfe8d

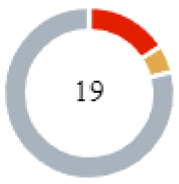
Audit Updates

Initial Audit	16 May 2023

Source Files

Filename	SHA256
Presale.sol	35e1b7a992bc0de86ce4f8c7da7f132450ddb7ee4454be9fa5e3598a91c79927

Findings Breakdown



🟥	Critical	3
🟧	Medium	1
●	Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
🟥 Critical	3	0	0	0
🟧 Medium	1	0	0	0
● Minor / Informative	15	0	0	0

Introduction

The Presale contract is designed to facilitate the sale of tokens during a presale event. It provides a platform for buyers to participate in the presale by purchasing tokens using either Ether or specific whitelisted tokens.

Functionality

Overview of Functionality:

- Presale Parameters: The contract allows the contract owner to set important parameters such as the presale start and end times, the total number of tokens available for sale, and the minimum and maximum buy limits per transaction.
- Token Whitelisting: The contract supports multiple whitelisted tokens, enabling buyers to purchase tokens from a variety of token options. Each token has its own predefined price in terms of the sale token.
- Token Price Calculation: The contract calculates the amount of sale tokens a buyer will receive based on the token price and the amount of Ether and the timestamp of the purchase of the presale tokens.
- Buy Limits: The contract enforces both a minimum and maximum buy limit per transaction. Buyers must adhere to these limits when participating in the presale.
- Bounce Levels: The contract includes the concept of bounce levels, which apply a bonus percentage to the purchased token amount based on the purchase size. Bounce levels encourage larger purchases by providing additional tokens to buyers.
- Token Distribution: Once the presale ends, buyers can withdraw their purchased tokens from the contract.
- Unlocking Mechanism: The contract implements a locking period for token withdrawals. This mechanism ensures that buyers cannot withdraw their tokens before a specified time.

Notes

The current contract implementation permits the owner to modify certain parameters and conduct another presale. It is important to note that the contract was not originally designed to host multiple presales. Consequently, conducting additional presales using the same contract may introduce complications or limitations. It is recommended to thoroughly assess the contract's business logic and limit the initiation of subsequent presales.

Buy Mechanism Architecture

The presale contract is designed to support a selling rate that exceeds a 1:1 ratio between the presale token and the whitelisted tokens or the native currency. This means that it can be configured to sell presale tokens at a ratio of 2:1 or even higher. However, the contract can not be configured to sell presale tokens at a ratio of 1:2 or any other ratio lower than that.

```
function getTokenAmount(
    address token,
    uint256 amount
) public view returns (uint256) {
    uint256 amtOut;
    uint tier = getCurrentTier();
    if (token != address(0)) {
        require(tokenWL[token] == true, "Presale: Token not
whitelisted");

        amtOut = amount.mul(10 ** saleTokenDec).div(
            tokenPrices[token][tier]
        );
    } else {
        amtOut = amount.mul(10 ** saleTokenDec).div(rate[tier]);
    }
    return amtOut;
}
```

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🔴	PTWM	Presale Tokens Withdrawal Mechanism	Unresolved
🔴	TTFPI	Total Tokens For Presale Inconsistency	Unresolved
🔴	ZD	Zero Division	Unresolved
🟠	TPI	Tier Percentages Inconsistency	Unresolved
●	RDS	Redundant Data Structure	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	BTMO	Buy Token Method Optimization	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L19	Stable Compiler Version	Unresolved

PTWM - Presale Tokens Withdrawal Mechanism

Criticality	Critical
Location	Presale.sol#L463,467
Status	Unresolved

Description

The contract owner has the authority to arbitrarily withdraw presale tokens before the presale is officially concluded. The withdrawal of tokens undermines the integrity of the presale process. This can have negative consequences, including disrupting the intended distribution mechanism and resulting in an unequal allocation of tokens. Additionally, it can affect the flow of transactions during the presale.

```
function withdraw(address token, uint256 amt) public onlyOwner
{
    IERC20(token).safeTransfer(msg.sender, amt);
}

function withdrawAll(address token) public onlyOwner {
    uint256 amt = IERC20(token).balanceOf(address(this));
    withdraw(token, amt);
}
```

Recommendation

It is recommended to implement a comprehensive and secure mechanism that enforces the proper conclusion of the presale before allowing the owner to withdraw the presale token. This mechanism should include a validation step to verify whether the presale has reached its intended end time or has met the required conditions before permitting any withdrawals.

TTFPI - Total Tokens For Presale Inconsistency

Criticality	Critical
Location	Presale.sol#L972
Status	Unresolved

Description

The owner has the authority to change the total presale token allocation even after the presale has started. As a result, inconsistency will be created between the distributed and the actual total presale tokens.

```
function setSaleTokenParams(
    address _saleToken,
    uint256 _totalTokensforSale
) external onlyOwner isPresaleStarted {
    saleToken = _saleToken;
    saleTokenDec = IERC20Metadata(saleToken).decimals();
    totalTokensforSale = _totalTokensforSale;
    IERC20(saleToken).safeTransferFrom(
        msg.sender,
        address(this),
        totalTokensforSale
    );
}
```

Recommendation

It is recommended to preconfigure the presale parameters prior to the presale start time and prevent mutation of core presale parameters after the presale starts.

ZD - Zero Division

Criticality	Critical
Location	Presale.sol#L1056
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

The `rate` and the `tokenPrices` could be set to zero.

```
function getTokenAmount(
    address token,
    uint256 amount
) public view returns (uint256) {
    ...
    if (token != address(0)) {
        ...
        amtOut = amount.mul(10 ** saleTokenDec).div(
            tokenPrices[token][tier]
        );
    } else {
        amtOut = amount.mul(10 **
saleTokenDec).div(rate[tier]);
    }
    return amtOut;
}
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow executing of the corresponding statements.

TPI - Tier Percentages Inconsistency

Criticality	Medium
Location	contracts/Presale.sol#L1025,1076
Status	Unresolved

Description

The contract configuration of the bounces is not properly sanitized. If the bounces `_amounts` are not sorted potential inconsistencies will be created in the tier percentages. Specifically, if the configured amounts are inserted in descending order, the largest amount will be selected, leading to the wrong tier selection.

```
function setBounces(
    uint256[] memory _amounts,
    uint256[] memory _percentages
) external onlyOwner {
    require(
        _amounts.length == _percentages.length,
        "Presale: Bounce arrays length mismatch"
    );
    delete bounces;
    for (uint256 i = 0; i < _amounts.length; i++) {
        bounces.push(Bounce(_amounts[i], _percentages[i]));
    }
}

for (uint256 i = 0; i < bounces.length; i++) {
    if (amount >= bounces[i].amount) {
        bounce = bounces[i].percentage;
    }
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications. The configured `_amounts` should be sorted in ascending order.

RDS - Redundant Data Structure

Criticality	Minor / Informative
Location	contracts/Presale.sol#L928,929
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 uses two data structures that contain duplicate information. The `buyersAmount` structure includes the `amount` variable, which corresponds to the `uint256` value found in the presaleData.

```
struct BuyerTokenDetails {
    uint amount;
    bool isClaimed;
}

mapping(address => BuyerTokenDetails) public buyersAmount;
mapping(address => uint256) public presaleData;
```

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 remove or modify one of the data structures.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Presale.sol#L979,1112
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
IERC20(saleToken).safeTransferFrom(
    msg.sender,
    address(this),
    totalTokensforSale
);
IERC20(_token).safeTransferFrom(msg.sender, address(this),
_amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance
Before Transfer
```

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	Presale.sol#L986,1025,1076
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 variables `presaleStartTime` and `presaleEndTime` are not properly sanitized.

```
function setPresaleTime(
    uint256 _presaleStartTime,
    uint256 _presaleEndTime
) external onlyOwner {
    presaleStartTime = _presaleStartTime;
    presaleEndTime = _presaleEndTime;
}
```

The `_percentages` are not properly sanitized.

```
function setBounces(
    uint256[] memory _amounts,
    uint256[] memory _percentages
) external onlyOwner {
    require(
        _amounts.length == _percentages.length,
        "Presale: Bounce arrays length mismatch"
    );
    delete bounces;
    for (uint256 i = 0; i < _amounts.length; i++) {
        bounces.push(Bounce(_amounts[i], _percentages[i]));
    }
}

for (uint256 i = 0; i < bounces.length; i++) {
    if (amount >= bounces[i].amount) {
        bounce = bounces[i].percentage;
    }
}
```

Recommendation

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

- The variable `presaleStartTime` should be lower than the `presaleEndTime` .
- The aggregation of the `_percentages` should not be greater than 1000.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	Presale.sol#L1007
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The method updateTokenRate configures the token price, not the rate.

```
function updateTokenRate(
    address _token,
    uint256[4] memory _price
) external onlyOwner {
    require(tokenWL[_token], "Presale: Token not whitelisted");
    tokenPrices[_token] = _price;
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	Presale.sol#L934
Status	Unresolved

Description

The contract uses the variable `totalTokensSold` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint256 public totalTokensSold;
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

BTMO - Buy Token Method Optimization

Criticality	Minor / Informative
Location	Presale.sol#L1075
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 `buyToken` method contains repetitive code segments. The function uses the same code segment in the if-statement.

```
if (_token != address(0)) {
    ...
    require(
        saleTokenAmt >= minBuyLimit,
        "Presale: Min buy limit not reached"
    );
    require(
        presaleData[msg.sender] + saleTokenAmt <= maxBuyLimit,
        "Presale: Max buy limit reached for this phase"
    );
    require(
        (totalTokensSold + saleTokenAmt) <= totalTokensforSale,
        "Presale: Total Token Sale Reached!"
    );

    IERC20(_token).safeTransferFrom(msg.sender, address(this),
_amount);
} else {
    ...
    require(
        saleTokenAmt >= minBuyLimit,
        "Presale: Min buy limit not reached"
    );
    require(
        presaleData[msg.sender] + saleTokenAmt <= maxBuyLimit,
        "Presale: Max buy limit reached for this phase"
    );
    require(
        (totalTokensSold + saleTokenAmt) <= totalTokensforSale,
        "Presale: Total Token Sale Reached!"
    );
}
```

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 duplicate code segment outside the if statement.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	Presale.sol#L972,995,1007,1015,1020,1085,1136
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of 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.

The contract does not emit events on the functions `addWhiteListedToken` , `updateTokenRate` , `setSaleTokenParams` , `buyToken` , and `withdrawToken`

```
function addWhiteListedToken(
    address _token,
    uint256[4] memory _price
) external onlyOwner {...}

function updateTokenRate(
    address _token,
    uint256[4] memory _price
) external onlyOwner {...}

function setSaleTokenParams(
    address _saleToken,
    uint256 _totalTokensforSale
) external onlyOwner isPresaleStarted {...}

function startUnlocking() external onlyOwner isPresaleEnded
{...}

function stopUnlocking() external onlyOwner isPresaleEnded
{...}

function buyToken(
    address _token,
    uint256 _amount
) external payable isPresaleStarted isPresaleNotEnded {..}

function withdrawToken() external {...}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. 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/Presale.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	contracts/Presale.sol#L973,974,987,988,996,997,1003,1008,1009,1026,1027,1086,1087,1148,1152
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.

```
address _saleToken
uint256 _totalTokensforSale
uint256 _presaleStartTime
uint256 _presaleEndTime
address _token
uint256[4] memory _price
uint256[4] memory _rate
uint256[] memory _amounts
uint256[] memory _percentages
uint256 _amount
uint _minBuyLimit
uint _maxBuyLimit
```

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/Presale.sol#L977,990,1149,1153
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.

```
saleTokenDec = IERC20Metadata(saleToken).decimals()
presaleStartTime = _presaleStartTime
minBuyLimit = _minBuyLimit
maxBuyLimit = _maxBuyLimit
```

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/Presale.sol#L538,569,601,645,663,680,698,782,800,816
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"
        );
    }

...
```

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/Presale.sol#L1063,1092,1138
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(tokenWL[token] == true, "Presale: Token not
whitelisted")
require(tokenWL[_token] == true, "Presale: Token not
whitelisted")

require(
            buyersAmount[msg.sender].isClaimed == false,
            "Presale: Already claimed"
        )
```

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/Presale.sol#L976
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.

```
saleToken = _saleToken
```

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/Presale.sol#L516,727
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 {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/Presale.sol#L9,253,347,373,453,487,742,868,895
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

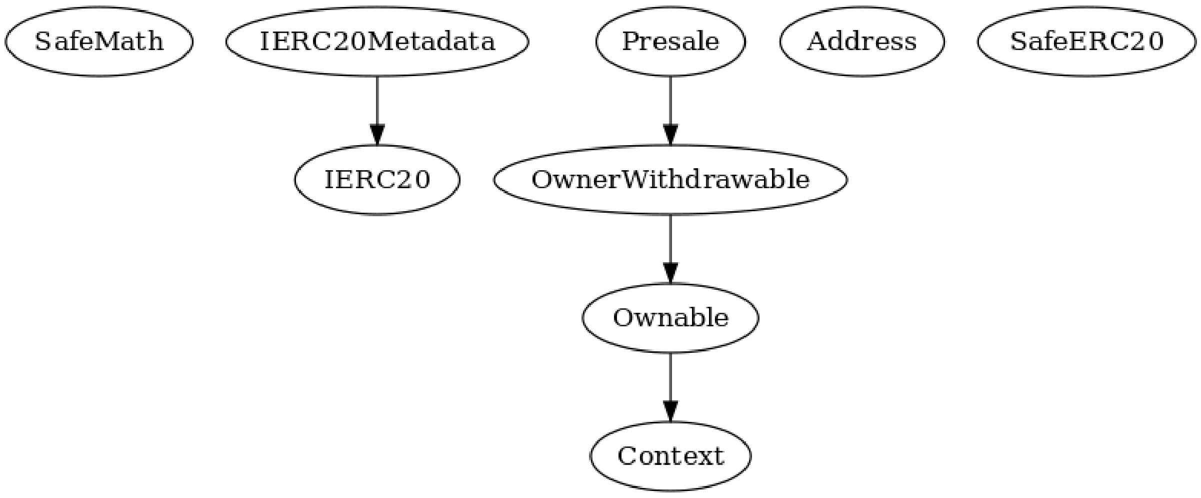
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-

	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
OwnerWithdrawable	Implementation	Ownable		
		External	Payable	-
		External	Payable	-
	withdraw	Public	✓	onlyOwner
	withdrawAll	Public	✓	onlyOwner
	withdrawCurrency	Public	✓	onlyOwner

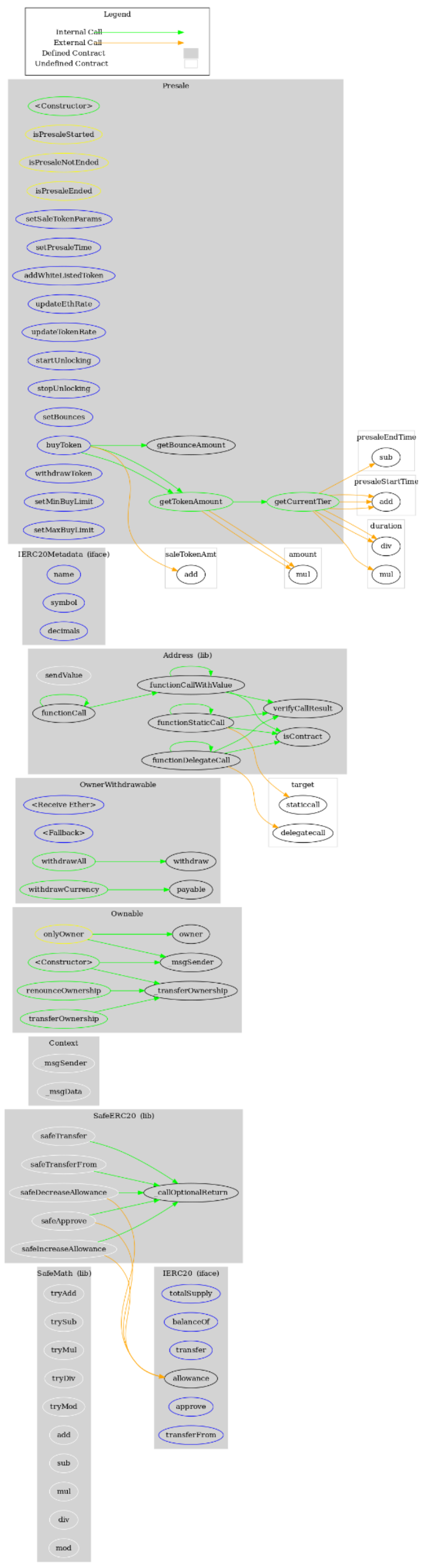
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-

	symbol	External		-
	decimals	External		-
Presale	Implementation	OwnerWithdrawable		
		Public	✓	-
	setSaleTokenParams	External	✓	onlyOwner isPresaleStarted
	setPresaleTime	External	✓	onlyOwner
	addWhiteListedToken	External	✓	onlyOwner
	updateEthRate	External	✓	onlyOwner
	updateTokenRate	External	✓	onlyOwner
	startUnlocking	External	✓	onlyOwner isPresaleEnded
	stopUnlocking	External	✓	onlyOwner isPresaleEnded
	setBounces	External	✓	onlyOwner
	getCurrentTier	Public		-
	getTokenAmount	Public		-
	getBounceAmount	Public		-
	buyToken	External	Payable	isPresaleStarted isPresaleNotEnded
	withdrawToken	External	✓	-
	setMinBuyLimit	External	✓	onlyOwner
	setMaxBuyLimit	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Presale
<Constructor>
isPresaleStarted
isPresaleNotEnded
isPresaleEnded
setSaleTokenParams
setPresaleTime
addWhiteListedToken
updateEthRate
updateTokenRate
startUnlocking
stopUnlocking
setBounces
buyToken
getBounceAmount
withdrawToken
getTokenAmount
getCurrentTier
setMinBuyLimit
setMaxBuyLimit
presaleEndTime
sub
presaleStartTime
add
duration
div
mul
IERC20Metadata (iface)
name
symbol
decimals
saleTokenAmt
add
amount
mul
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
OwnerWithdrawable
<Receive Ether>
<Fallback>
withdrawAll
withdraw
withdrawCurrency
payable
Ownable
onlyOwner
owner
<Constructor>
msgSender
renounceOwnership
transferOwnership
transferOwnership
Context
msgSender
_msgData
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeDecreaseAllowance
callOptionalReturn
safeApprove
safeIncreaseAllowance
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

INTDESTCOIN contract implements a financial mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io