Cyberscope

Audit Report

AI MASA

Aug 2023

Network	BSC
Address	0x6e773D3F2B3016Ff5D89C4B600776FB8F3B75399
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **2**
Audit Updates 2
Source Files 2
Overview **3**
Functionality 3
Findings Breakdown **5**
Diagnostics **6**
UC - Unreachable Code 8
Description 8
Recommendation 8
US - Untrusted Source 10
Description 10
Recommendation 10
OCTD - Transfers Contract's Tokens 11
Description 11
Recommendation 11
RCC - Redundant Condition Check 12
Description 12
Recommendation 12
DKO - Delete Keyword Optimization 13
Description 13
Recommendation 13
PTAI - Potential Transfer Amount Inconsistency 14
Description 14
Recommendation 14
MEE - Missing Events Emission 16
Description 16
Recommendation 17
AOI - Arithmetic Operations Inconsistency 18
Description 18
Recommendation 18
MC - Missing Check 19
Description 19
Recommendation 19
PBV - Percentage Boundaries Validation 20
Description 20
Recommendation 21
PVI - Presale Variables Inconsistency 22

Description 22
Recommendation 23
PII - Presale Initialization Inconsistency 24
Description 24
Recommendation 24
RC - Redundant Calculations 26
Description 26
Recommendation 26
CR - Code Repetition 27
Description 27
Recommendation 28
DPI - Decimals Precision Inconsistency 29
Description 29
Recommendation 30
RCF - Reusable Code Functionality 31
Description 31
Recommendation 31
RPC - Redundant Precision Calculation 32
Description 32
Recommendation 32
RSML - Redundant SafeMath Library 33
Description 33
Recommendation 33
RSK - Redundant Storage Keyword 34
Description 34
Recommendation 34
L04 - Conformance to Solidity Naming Conventions 35
Description 35
Recommendation 36
L06 - Missing Events Access Control 37
Description 37
Recommendation 37
L07 - Missing Events Arithmetic 38
Description 38
Recommendation 38
L13 - Divide before Multiply Operation 39
Description 39
Recommendation 39
L16 - Validate Variable Setters 40
Description 40
Recommendation 40
L19 - Stable Compiler Version 41

Description 41
Recommendation 41
L20 - Succeeded Transfer Check 42
Description 42
Recommendation 42
Functions Analysis 43
Inheritance Graph 46
Flow Graph 47
Summary 48
Disclaimer 49
About Cyberscope 50

Review

Contract Name	PreSale
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6e773d3f2b3016ff5d89c4b600776fb8f3b75399
Address	0x6e773d3f2b3016ff5d89c4b600776fb8f3b75399
Network	BSC

Audit Updates

Initial Audit	19 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/masa/v1/audit.pdf
Corrected Phase 2	28 Jul 2023https://github.com/cyberscope-io/audits/blob/main/masa/v2/audit.pdf
Corrected Phase 3	03 Aug 2023

Source Files

Filename	SHA256
PreSale.sol	2959a1b619367d007b397d35980a2ec322298240efd2d190c1979b2d5c38cd8d

Overview

The Presale contract is designed to manage the pre-sale of a specific BEP20 token in exchange for USDT. The contract provides mechanisms for users to buy tokens during the pre-sale period, claim their purchased tokens after the pre-sale, and receive airdrops. The contract also incorporates a referral system, allowing users to earn rewards for referring other participants.

Out of Scope Address: The contract initializes the `priceFeedBnb` with an instance of `AggregatorV3Interface` using the address `0x0567F2323251f0Aab15c8dFb1967E4e8A7D42aeE`. It's important to note that the behavior, functionality, and security of the contract or code at this address are out of the scope of this contract audit. Any interactions with this address or reliance on its data should be treated with caution, and it's recommended to conduct a separate audit or review for the contract deployed at that address.

Functionality

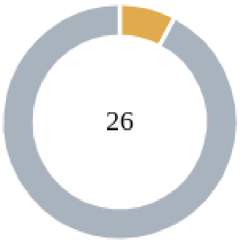
Overview of Functionality:

- Presale Starting Parameters: The contract allows the owner to set in the constructor various parameters, such as the reward percentages, the airdrop amount, the minimum and maximum USDT amounts for buying tokens, the hard cap for tokens and USDT, and the end time for the pre-sale.
- Token Conversion: The contract provides a mechanism (`usdtToToken` function) to convert the USDT amount to its equivalent in the sale token, ensuring users receive the correct number of tokens for their USDT. Also the `bnbToToken` function calculates the equivalent number of sale tokens for a given BNB amount and the `bnbToUsdt` function directly converts a BNB amount to its equivalent in USDT using the real-time price fetched from `getLatestPriceBnb`.
- Referral System: The `setReferrer` function allows users to set a referrer, and the referrer can earn rewards based on the amount purchased by the user. The reward percentage for referrals can be set by the contract owner.
- Token Purchase: Users can buy tokens using the `buyToken` function by specifying the amount of USDT they wish to spend and their referrer's address. The function ensures that the purchase does not exceed the hard caps and that the

pre-sale is still ongoing. Similarly, with the `buyWithBNB` function, users can send BNB directly to the contract to purchase tokens, also specifying a referrer's address. This function calculates the token equivalent for the sent BNB, checks purchase limits, and updates both the total BNB raised and the buyer's claimable tokens. Both functions log the purchase details accordingly.

- Token Claiming: After the pre-sale period, users can claim their purchased tokens using the `claim` function. This function transfers the tokens to the user and resets their claimable amount.
- Airdrop Mechanism: The contract includes an `airDrop` function that allows users to claim the airdrop tokens. If a user has a referrer, the referrer also receives a percentage of the airdrop.
- Owner Controls: The contract owner has various controls, such as transferring funds, and changing ownership. Also the owner has the authority to change the pre-sale parameters, such as airdrop percentages, token price, hard caps, and referral percentages. These controls ensure that the owner can manage the pre-sale effectively and make necessary adjustments.

Findings Breakdown



Critical	0
Medium	2
Minor / Informative	24

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	2	0	0	0
Minor / Informative	24	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	UC	Unreachable Code	Unresolved
●	US	Untrusted Source	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	RCC	Redundant Condition Check	Unresolved
●	DKO	Delete Keyword Optimization	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	MC	Missing Check	Unresolved
●	PBV	Percentage Boundaries Validation	Unresolved
●	PVI	Presale Variables Inconsistency	Unresolved
●	PII	Presale Initialization Inconsistency	Unresolved
●	RC	Redundant Calculations	Unresolved
●	CR	Code Repetition	Unresolved

●	DPI	Decimals Precision Inconsistency	Unresolved
●	RCF	Reusable Code Functionality	Unresolved
●	RPC	Redundant Precision Calculation	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

UC - Unreachable Code

Criticality	Medium
Location	PreSale.sol#L126
Status	Unresolved

Description

The contract is facilitating token purchases through the `buyToken` function. Within this function, the user's existence is determined by the `user.isExists` boolean flag. Initially, the code checks `if(!user.isExists)` is true and, if so, sets `user.isExists` to true. Subsequently, there's another identical check. Due to the initial assignment, the second check (`if(!user.isExists)`) will always evaluate as false. Consequently, the subsequent code block, which includes the incrementation of the `totalBuyer` variable, will never be executed.

Additionally, it's worth noting that the `totalBuyer` variable is initialized as private, making it inaccessible for external queries or interactions.

```
uint256 totalBuyer;

function buyToken(uint256 _amount, address _referrer) public {
        ...
        if (!user.isExists) {
            user.isExists = true;
        }
        ...
        if (!user.isExists) {
            user.isExists = true;
            totalBuyer++;
        }
        ...
    }
```

Recommendation

It is recommended to streamline the logic by removing the redundant check for `user.isExists`. The incrementation of the `totalBuyer` variable should be placed within the initial check where `user.isExists` is set to true. This adjustment ensures that the `totalBuyer` count is accurately updated for each new user.

Additionally, If the intention behind the totalBuyer variable is to provide information on the total number of buyers, it would be beneficial to make it public. This would allow external parties to query and obtain data on the total number of buyers, enhancing transparency and usability of the contract.

US - Untrusted Source

Criticality	Medium
Location	PreSale.sol#L326
Status	Unresolved

Description

The contract is designed to interact with external contracts, specifically through the `USDT` , `MASA` , and `priceFeedBnb` variables. These external contracts play a crucial role in determining the transaction's flow within the Presale contract. However, the `changeAddresses` function allows the owner to modify these addresses, potentially pointing them to untrusted or malicious contracts. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
    function changeAddresses(
        address _usdt,
        address _masa,
        address _aggregator
    ) public onlyOwner {
        USDT = IBEP20(_usdt);
        MASA = IBEP20(_masa);
        priceFeedBnb = AggregatorV3Interface(_aggregator);
    }
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	PreSale.sol#L371
Status	Unresolved

Description

The contract is designed to utilize the `MASA` token as part of its reward mechanism. However, the `transferStuckFunds` function, grants the contract owner the authority to claim any amount of the `MASA` token balance. This poses a significant risk as the owner can potentially drain all the `MASA` tokens from the contract, leaving it without sufficient balance to fulfill its intended reward distributions or other obligations.

```
    function transferStuckFunds(uint256 _value)
        external
        onlyOwner
        returns (bool)
    {
        MASA.transfer(owner, _value);
        return true;
    }
```

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.

RCC - Redundant Condition Check

Criticality	Minor / Informative
Location	PreSale.sol#L241
Status	Unresolved

Description

The contract is using redundant conditional checks in the `setReferrer` function. Specifically, after checking `if user.referrer == address(0)`, the function then checks `if (user.referrer != address(0))`. Given the binary nature of this condition (either the referrer is the zero address or it isn't), the second check is redundant and can be simplified using the `else` keyword. This not only makes the code cleaner and more readable but also slightly optimizes gas usage by eliminating the unnecessary condition check.

```
function setReferrer(
        address _user,
        address _referrer,
        uint256 _amount,
        bool _val
    ) internal {
        UserInfo storage user = users[_user];
        if (user.referrer == address(0)) {
        ...
        if (user.referrer != address(0)) {
        ...
    }
```

Recommendation

It is recommended to replace the redundant `if (user.referrer != address(0))` check with the `else` keyword to streamline the code and improve clarity.

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	PreSale.sol#L209
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
    user.claimAbleAmount = 0;
     user.referrerReward = 0;
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L210,213
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

```
   MASA.transferFrom(owner, msg.sender, user.referrerReward);
    ...
    MASA.transferFrom(owner, msg.sender, user.claimAbleAmount);
```

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`

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	PreSale.sol#L203,248,267,271,279
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.

```
    function claim() public {
        UserInfo storage user = users[msg.sender];
        require(user.isExists, "Didn't bought");
        require(block.timestamp >= preSaleTime, "Wait for the PreSale
endtime");
        token.transfer(msg.sender, user.referrerReward);
        token.transfer(msg.sender, user.claimAbleAmount);
        user.claimAbleAmount = 0;
        user.referrerReward = 0;
    }
    function airDrop() external {
        UserInfo storage user = users[msg.sender];
        ...
    }

    function changeAirDropAmount(uint256 _amount) external onlyOwner {
        airDropAmount = _amount * 10**token.decimals();
    }

    function setPreSaleAmount(uint256 _minAmount, uint256 _maxAmount)
        external
        onlyOwner
    {
        minAmount = _minAmount;
        maxAmount = _maxAmount;
    }

    function setpreSaleTime(uint256 _time) external onlyOwner {
        preSaleTime = _time;
    }
```

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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L141,153
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
numberOfTokens + soldToken <= tokenHardCap &&
...
soldToken = soldToken.add(numberOfTokens);
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

MC - Missing Check

Criticality	Minor / Informative
Location	PreSale.sol#L285,293
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.

Specifically the `minAmount` should be less than the `maxAmount` when set by the `setPreSaleAmount` function.

Also the variable `preSaleTime` should be greater than the current timestamp.

```
function setPreSaleAmount(uint256 _minAmount, uint256 _maxAmount)
        external
        onlyOwner
    {
        minAmount = _minAmount;
        maxAmount = _maxAmount;
    }
```

```
function setpreSaleTime(uint256 _time) external onlyOwner {
        preSaleTime = _time;
    }
```

Recommendation

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

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	PreSale.sol#L243,267
Status	Unresolved

Description

The contract is using the variables `airDropRefPercentage` and `referrerPercentage` for calculations. However, these variables are used in multiplication operations and if `airDropRefPercentage` or `referrerPercentage` is set to a value greater than `100`, it could lead to incorrect calculations, potentially causing unintended behavior or financial discrepancies within the contract's operations.

```
function setReferrer(
        address _user,
        address _referrer,
        uint256 _amount,
        bool _val
    ) internal {
        ...
            users[user.referrer].referrerReward += usdtToToken(
                (_amount * referrerPercentage) / percentageDivider
        ...
        }

    function airDrop() external {
        ...
            IBEP20(MASA).transferFrom(
                owner,
                user.referrer,
                (airDropAmount * airDropRefPercentage) /
percentageDivider
        ...
    }
```

Recommendation

It is recommended to ensure that the values of `airDropRefPercentage` and `referrerPercentage` cannot exceed `100`. This can be achieved by adding checks whenever these variables are set.

PVI - Presale Variables Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L379
Status	Unresolved

Description

The contract contains the `setNewRound` function, that grants the owner the authority to modify key parameters of the presale, including the total token allocation (`_tokenHardCap`). As a result, this function can be invoked even after the presale has commenced.

This capability introduces a potential risk where the owner can unilaterally alter the presale's dynamics, leading to inconsistencies between the initial promises or expectations set for users and the actual token allocations.

Additionaly, the owner can modify critical variables, such as the `_tokenHardCap` , `_UsdtHardCap` , and `_tokenPerUsd` . However, these variables are not being properly checked for their proper shape before processing. This lack of validation can introduce vulnerabilities, especially when these variables determine the behavior and outcomes of the presale. The unchecked authority of the contract owner to change these variables without any preconditions or validations can lead to potential misuse and unintended consequences.

```
    function setNewRound(
        uint256 _tokenPerUsd,
        uint256 _preSaleTime,
        uint256 _soldToken,
        uint256 _tokenHardCap,
        uint256 _UsdtHardCap,
        uint256 _amountRaised
    ) public onlyOwner {
        tokenPerUsd = _tokenPerUsd;
        preSaleTime = _preSaleTime;
        soldToken = _soldToken;
        tokenHardCap = _tokenHardCap;
        UsdtHardCap = _UsdtHardCap;
        amountRaised = _amountRaised;
    }
}
```

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. This can be achieved by adding a condition within the `setNewRound` function to check if the current timestamp is before the presale start time. By doing so, the contract will ensure that once the presale begins, the core parameters remain immutable.

Additionaly, it is recommended to introduce additional checks in the `setNewRound` function to ensure the integrity and consistency of the variables being modified. Specifically, when updating any two of the variables `_tokenHardCap`, `_UsdtHardCap`, and `_price`, the third variable should be automatically calculated based on: `_tokenHardCap = _UsdtHardCap * _tokenPerUsd`. This approach ensures that the values remain consistent with each other and reduces the risk of manual errors or potential manipulation. By allowing only two out of the three variables to be set directly and computing the third, the contract can maintain a more predictable and secure state.

PII - Presale Initialization Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L111
Status	Unresolved

Description

The contract is initializing various parameters within its constructor. However, there is no mechanism in place to ensure that the required amount of tokens for the presale is transferred to the contract or to validate that the token balance of the contract is sufficient to meet the demands of the presale. This oversight could lead to potential issues during the presale, where users might not receive their expected tokens after contributing funds.

```
      constructor() {
        owner = payable(msg.sender);
        MASA = IBEP20(0x6730C6B035cD9bF185cd86d74204F30b3F260130);
        USDT = IBEP20(0x55d398326f99059fF775485246999027B3197955);
        priceFeedBnb =
AggregatorV3Interface(0x0567F2323251f0Aab15c8dFb1967E4e8A7D42aeE);
        referrerPercentage = 10_00;
        airDropRefPercentage = 0;
        percentageDivider = 100_00;
        airDropAmount = 0 * 10**MASA.decimals();
        tokenPerUsd = 10;
        UsdtHardCap = 10000000 * 10**USDT.decimals();
        tokenHardCap = 100000000 * 10**MASA.decimals();
        minAmount = 500 * 10**MASA.decimals();
        maxAmount = 50000000 * 10**MASA.decimals();
        preSaleTime = block.timestamp + 300 days;
    }
```

Recommendation

It is recommended to implement a mechanism during the presale initialization that either automatically transfers the required amount of tokens to the contract, or, if the tokens are expected to be transferred to the contract manually, add a validation function that checks the token balance of the contract to ensure that there are enough tokens in the contract before the presale starts. This validation should be executed before any user is allowed to participate in the presale.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	PreSale.sol#L230,238
Status	Unresolved

Description

The contract contains the `setReferrer` function that aims to assign a referrer to a user under certain conditions. Within this function, there's a check to determine if the user already has a referrer using the condition `if (user.referrer == address(0))`. If the user doesn't have a referrer, the function proceeds to evaluate further conditions to potentially assign one. However, in the scenario where the conditions for setting a valid referrer are not met, the function redundantly sets `user.referrer` to the zero address `(address(0))` again. Given that the outer condition has already established that `user.referrer` is the zero address, this assignment is unnecessary and introduces redundant code in the contract.

```
if (user.referrer == address(0)) {
        if (…) {
                ...
        } else {
                user.referrer = address(0);
        }
    }
```

Recommendation

It is recommended to remove the line `user.referrer = address(0);` from the `else` block. This will streamline the function, reduce gas costs slightly, and improve the clarity of the code by avoiding redundant operations.

CR - Code Repetition

Criticality	Minor / Informative
Location	PreSale.sol#L126,156
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specifically, the contract is using with two functions, `buyToken` and `buyWithBNB` , which share a significant amount of similar functionality. Both functions handle the purchase of tokens, but they differ only in the payment method: `buyToken` uses USDT, while `buyWithBNB` uses BNB. The majority of the logic, including user existence checks, token amount calculations, and hard cap validations, is duplicated across both functions.

```
function buyToken(uint256 _amount, address _referrer) public {
        UserInfo storage user = users[msg.sender];
        setReferrer(msg.sender, _referrer, _amount, true);
        if (!user.isExists) {
            user.isExists = true;
        }
        uint256 numberOfTokens = usdtToToken(_amount);

        require(
            numberOfTokens >= minAmount && numberOfTokens <= maxAmount,
            "BEP20: Amount not correct"
        );
        require(
            numberOfTokens + soldToken <= tokenHardCap &&
                _amount + amountRaised <= UsdtHardCap,
            "Exceeding HardCap"
        );
        require(block.timestamp < preSaleTime, "BEP20: PreSale over");
        if (!user.isExists) {
            user.isExists = true;
            totalBuyer++;
        }
        USDT.transferFrom(msg.sender, address(this), _amount);
        amountRaised += _amount;
        user.claimAbleAmount += numberOfTokens;
        soldToken = soldToken.add(numberOfTokens);
        emit BuyToken(msg.sender, usdtToToken(_amount));
    }

    // to buy AI MASA token during preSale time => for web3 use
    function buyWithBNB(address _referrer) public payable {
        ...
        setReferrer(msg.sender, _referrer, msg.value, false);
        ...
    }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L196
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

Specifically, the contract using the `getLatestPriceBnb` function that retrieves the latest price data for BNB. This function fetches the price from the `priceFeedBnb` oracle and then divides it by `1e8` to adjust for decimals. However, there is a potential inconsistency in the way the decimals are handled. The specification does not strictly define the implementation of the decimals field, leading to variations in its precision across different contracts. The `latestRoundData` function from the oracle returns the price with its own specific decimal precision, which may not always be `1e8`. If the oracle's decimal precision differs from the hardcoded `1e8` value, the result of the `getLatestPriceBnb` function may not accurately represent the intended price.

```
    function getLatestPriceBnb() public view returns (uint256) {
        (, int256 price, , , ) = priceFeedBnb.latestRoundData();
        return uint256(price).div(1e8);
    }
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the token with the greatest digits.

It is recommended to use each time the decimals which the `latestRoundData` returns. Instead of hardcoding the division by `1e8`, dynamically fetch the decimal precision from the oracle or the relevant data source and use that for the calculation. This approach ensures that the price is always adjusted correctly, regardless of the decimal precision used by the oracle, enhancing the accuracy and reliability of the `getLatestPriceBnb` function.

RCF - Reusable Code Functionality

Criticality	Minor / Informative
Location	PreSale.sol#L182
Status	Unresolved

Description

The contract is employing the `bnbToToken` function to convert a specified BNB amount to its token equivalent. Within this function, there is a calculation to convert BNB to its USD equivalent, denoted as `bnbToUsd`. However, the contract also contains a separate function named `bnbToUsdt` which, presumably performs the BNB to USDT conversion. This suggests a redundancy in the code, as the same conversion logic might be implemented in two different places. By not utilizing the `bnbToUsdt` function within the `bnbToToken` function, the contract is missing an opportunity to streamline its code and ensure consistent conversion logic.

```
    function bnbToToken(uint256 _amount) public view returns (uint256) {
        uint256 precision = 1e4;
        uint256 bnbToUsd =
precision.mul(_amount).mul(getLatestPriceBnb()).div(
            1e18
        );
        uint256 numberOfTokens = bnbToUsd.mul(tokenPerUsd);
        return numberOfTokens.mul(10**token.decimals()).div(precision);
    }
```

Recommendation

It is recommended to refactor the `bnbToToken` function to use the `bnbToUsdt` function for the BNB to USDT conversion. This will not only simplify the code by eliminating redundancy but also ensure that any updates or fixes made to the BNB to USDT conversion logic in the future will be consistently applied across the contract. This approach promotes code reusability and maintainability, leading to a more robust and efficient smart contract.

RPC - Redundant Precision Calculation

Criticality	Minor / Informative
Location	PreSale.sol#L182
Status	Unresolved

Description

The contract is utilizing a redundant precision calculation in the `bnbToToken` function. Specifically, the contract defines a `precision` variable with a value of `1e4`. This precision value does not offer any additional accuracy in the calculations, as it is merely a standard precision. Furthermore, the division by `1e18` in the calculation of `bnbToUsd` could be more efficiently integrated into the final calculation of the `numberOfTokens`.

```
    function bnbToToken(uint256 _amount) public view returns (uint256) {
         uint256 precision = 1e4;
         uint256 bnbToUsd =
precision.mul(_amount).mul(getLatestPriceBnb()).div(
             1e18
         );
         uint256 numberOfTokens = bnbToUsd.mul(tokenPerUsd);
         return numberOfTokens.mul(10**token.decimals()).div(precision);
    }
```

Recommendation

It is recommended to simplify the calculation by removing the unnecessary `precision` variable and its associated operations. The division by `1e18` can be incorporated into the final calculation of `numberOfTokens` to streamline the function and reduce computational overhead. This will not only make the code more readable but also optimize gas usage.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	PreSale.sol#L359
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
UserInfo storage user
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PreSale.sol#L68,69,87,124,154,180,197,220,221,222,223,250,273,277,281,289,294,299,304,313,323,324,325,348,368,369,376,377,378,379,380,381
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.

```
IBEP20 public MASA
IBEP20 public USDT
uint256 public UsdtHardCap
uint256 _amount
address _referrer
uint256 _value
address _user
bool _val
uint256 _tokenPerUsd
uint256 _minAmount
uint256 _maxAmount
uint256 _time
address payable _newOwner
address _usdt

...
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	PreSale.sol#L295
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
owner = _newOwner
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	PreSale.sol#L274,278,290,371,383
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.

```
tokenPerUsd = _tokenPerUsd
airDropAmount = _amount * 10**MASA.decimals()
preSaleTime = _time
airDropRefPercentage = _airDropRefPercentage
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	PreSale.sol#L182,185,251,254
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.

```
uint256 numberOfTokens = _amount.mul(tokenPerUsd).div(
            10**USDT.decimals()
        )
return numberOfTokens.mul(10**MASA.decimals())
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PreSale.sol#L295
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.

```
owner = _newOwner
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	PreSale.sol#L2
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	PreSale.sol#L146,210,213,261,263,309,318
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.

```
USDT.transferFrom(msg.sender, address(this), _amount)
MASA.transferFrom(owner, msg.sender, user.referrerReward)
MASA.transferFrom(owner, msg.sender, user.claimAbleAmount)
IBEP20(MASA).transferFrom(owner, msg.sender, airDropAmount)

IBEP20(MASA).transferFrom(
                owner,
                user.referrer,
                (airDropAmount * airDropRefPercentage) /
percentageDivider
            )
USDT.transfer(owner, _value)
MASA.transfer(owner, _value)
```

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.

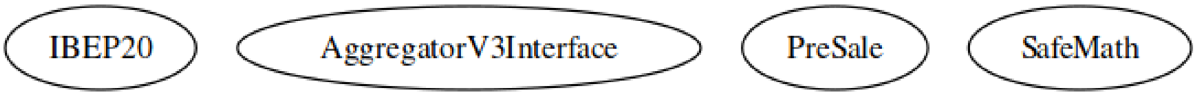
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
AggregatorV3Interface	Interface			
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-
PreSale	Implementation			
		Public	✓	-

	buyToken	Public	✓	-
	buyWithBNB	Public	Payable	-
	bnbToToken	Public		-
		External	Payable	-
	getLatestPriceBnb	Public		-
	bnbToUsdt	Public		-
	claim	Public	✓	-
	setReferrer	Internal	✓	
	usdtToToken	Public		-
	airDrop	External	✓	-
	changePrice	External	✓	onlyOwner
	changeAirDropAmount	External	✓	onlyOwner
	setPreSaleAmount	External	✓	onlyOwner
	setpreSaleTime	External	✓	onlyOwner
	changeOwner	External	✓	onlyOwner
	transferFunds	External	✓	onlyOwner
	transferUSDTFunds	External	✓	onlyOwner
	transferStuckFunds	External	✓	onlyOwner
	changeAddresses	Public	✓	onlyOwner
	totalSupply	External		-
	getCurrentTime	Public		-
	contractBalanceBnb	External		-
	getContractTokenBalance	External		-

	getUserInfo	Public		-
	setReferrerPercentage	Public	✓	onlyOwner
	setNewRound	Public	✓	onlyOwner
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
mod
PreSale
onlyOwner
buyWithBNB
buyToken
<Constructor>
<Receive Ether>
claim
airDrop
changePrice
changeAirDropAmount
setPreSaleAmount
setpreSaleTime
changeOwner
transferFunds
transferStuckFunds
transferUSDTFunds
changeAddresses
totalSupply
getCurrentTime
contractBalanceBnb
getContractTokenBalance
getUserInfo
setReferrerPercentage
setNewRound
bnbToUsdt
setReferrer
payable
getLatestPriceBnb
bnbToToken
usdtToToken
AggregatorV3Interface (iface)
decimals
description
version
getRoundData
latestRoundData
precision
mul
bnbToUsd
mul
numberOfTokens
mul
_amount
mul
soldToken
add
owner
transfer
IBEP20 (iface)
decimals
transferFrom
balanceOf
transfer
totalSupply
approve
allowance

Summary

AI MASA Presale contract implements a financial 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