Audit Report
AI MASA

May 2023

Network BSC Testnet

Address 0xbd4cfC9B4A3AC2947A51eA600daF13afafF41579

Audited by © cyberscope

Table of Contents

Review

Contract Name	PreSale
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0xbd4cfc9b4a3ac2947a51ea600daf13afaff41579
Address	0xbd4cfc9b4a3ac2947a51ea600daf13afaff41579
Network	BSC_TESTNET

Audit Updates

Initial Audit	19 Jul 2023

Source Files

Filename	SHA256
PreSale.sol	83c93798a46e2e2233a6b7773f3624e502dc61ae56da15054a7e711510 59b746

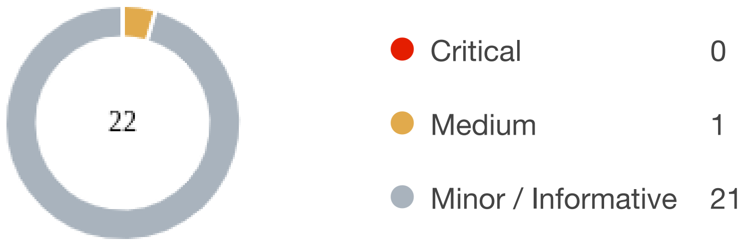
Overview

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.

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.
- 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.
- 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



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	21	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	IVA	Inconsistent Variable Amounts	Unresolved
●	RC	Redundant Calculations	Unresolved
●	PII	Presale Initialization Inconsistency	Unresolved
●	ZD	Zero Division	Unresolved
●	PVI	Presale Variables Inconsistency	Unresolved
●	IUA	Incorrect USDT Address	Unresolved
●	PBV	Percentage Boundaries Validation	Unresolved
●	MC	Missing Check	Unresolved
●	DKO	Delete Keyword Optimization	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	AOI	Arithmetic Operations Inconsistency	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
●	L22	Potential Locked Ether	Unresolved

IVA - Inconsistent Variable Amounts

Criticality	Medium
Location	PreSale.sol#L117,131,142
Status	Unresolved

Description

The contract contains an inconsistency in the representation and calculation of the amounts between the `referrerReward` and `claimAbleAmount` variables. Specifically, within the `setReferrer` function, the `referrerReward` is derived directly from the USDT `_amount`, implying a direct correlation between the USDT value and the reward. On the other hand, inside the `buyToken` function, the `claimAbleAmount` is calculated based on the token equivalent of the USDT _amount, using the `usdtToToken` conversion function. This discrepancy in calculation methodologies can lead to potential miscalculations, and unintended behaviors, since both the variables `referrerReward` and `claimAbleAmount` are used as the amount of tokens to be transferred to users in the `claim` function.

```
    function buyToken(uint256 _amount, address _referrer) public {
        ...
        uint256 numberOfTokens = usdtToToken(_amount);

        ...
        user.claimAbleAmount += numberOfTokens;
        ...
    }

        function setReferrer(address _user, address _referrer, uint256
_amount) internal {
        ...
            users[_referrer].referrerReward += _amount *
referrerPercentage/percentageDivider;
        ...
        }

    function claim() public {
        ...
        token.transfer(msg.sender, user.referrerReward);
        token.transfer(msg.sender, user.claimAbleAmount);
        ...
    }
```

Recommendation

It is recommended to standardize the representation and calculation of both `referrerReward` and `claimAbleAmount` to ensure they consistently represent token amounts.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	PreSale.sol#L126
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.

PII - Presale Initialization Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L80
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(0xBA02934d2DD50445Fd08E975eDE02CA6C609d4db);
      token = IBEP20(0xB4459a0EABd82BFAeA6Ea38192556FB0BE8dDf10);
      USDT = IBEP20(0xB4459a0EABd82BFAeA6Ea38192556FB0BE8dDf10);
      referrerPercentage = 7_00;
      airDropRefPercentage = 5_00;
      percentageDivider = 100_00;
      airDropAmount = 100 * 10**token.decimals();
      tokenPerUsd = 100;
      UsdtHardCap = 100000 * 10**USDT.decimals();
      tokenHardCap = 10000000 * 10**token.decimals();
      minAmount = 10 * 10**USDT.decimals();
      maxAmount = 100 * 10**USDT.decimals();
      preSaleTime = block.timestamp + 1 hours;
  }
```

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.

ZD - Zero Division

Criticality	Minor / Informative
Location	PreSale.sol#L131,163,212
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. Specifically, the `percentageDivider` variable can change value through the `changeValues` function. The `changeValues` function permits the `percentageDivider` to be set to zero. Setting the `percentageDivider` to zero can lead to a division by zero error. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
users[_referrer].referrerReward += _amount *
referrerPercentage/percentageDivider;
...

IBEP20(token).transfer(
                user.referrer,
                (airDropAmount * airDropRefPercentage) /
percentageDivider
            );
...

    function changeValues(
        uint256 _airDropRefPercentage,
        uint256 _percentageDivider,
        uint256 _price,
        uint256 _soldToken,
        uint256 _tokenHardCap,
        uint256 _UsdtHardCap,
        uint256 _amountRaised,
        uint256 _referrerPercentage
    ) public onlyOwner {
        airDropRefPercentage = _airDropRefPercentage;
        referrerPercentage = _referrerPercentage;
        percentageDivider = _percentageDivider;
        ...
    }
```

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 the execution of the corresponding statements.

PVI - Presale Variables Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L212
Status	Unresolved

Description

The contract contains the `changeValues` 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 `_price` . However, these variables are not being properly checked for their proper value before setting. 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 changeValues(
        uint256 _airDropRefPercentage,
        uint256 _percentageDivider,
        uint256 _price,
        uint256 _soldToken,
        uint256 _tokenHardCap,
        uint256 _UsdtHardCap,
        uint256 _amountRaised,
        uint256 _referrerPercentage
    ) public onlyOwner {
        airDropRefPercentage = _airDropRefPercentage;
        referrerPercentage = _referrerPercentage;
        percentageDivider = _percentageDivider;
        tokenPerUsd = _price;
        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 `changeValues` 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 `changeValues` 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 * _price`. 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.

IUA - Incorrect USDT Address

Criticality	Minor / Informative
Location	PreSale.sol#L84
Status	Unresolved

Description

The contract is currently initialized with an address for the `USDT` token that does not match the official `USDT` address on the BEP20 (Binance Smart Chain) network. Using an incorrect address can lead to a variety of issues, including the inability to correctly interact with the USDT token, potential loss of funds, and confusion for users.

```
USDT = IBEP20(0xB4459a0EABd82BFAeA6Ea38192556FB0BE8dDf10);
```

Recommendation

It is recommended that the team verify and update the smart contract to include the correct address of the USDT token on the BEP20 network. Ensuring the accuracy of token addresses is crucial for the smooth operation of the contract and to prevent potential issues or vulnerabilities. If the presale is planned to be on the BEP20 network, it's imperative to use the correct `USDT` address, which is `0x55d398326f99059ff775485246999027b3197955` at the time of the report.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	PreSale.sol#L107
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. Specifically, the comment `BEP20: Amount not correct` suggests that there is an issue related to a `BEP20` token amount. However, in the function `buyToken` the variable `_amount` is used to deposit `USDT` tokens and not `BEP20` tokens. As a result, the users will not be able to find the root cause of the error.

```
function buyToken(uint256 _amount, address _referrer) public {
        ...
        require(
            _amount >= minAmount && _amount <= maxAmount,
            "BEP20: Amount not correct"
        );
        ...
        USDT.transferFrom(msg.sender, address(this), _amount);
        ...
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	PreSale.sol#L122,155
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) internal {
        ...
            users[_referrer].referrerReward += _amount *
referrerPercentage/percentageDivider;
        ...
        }

    function airDrop() external {
        ...
            IBEP20(token).transfer(
                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.

MC - Missing Check

Criticality	Minor / Informative
Location	PreSale.sol#L173,181
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.

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	PreSale.sol#L144,145
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#L142,143
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

```
   token.transfer(msg.sender, user.referrerReward);
    token.transfer(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#L138,155,169,173,181
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#L110,118
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.

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#L240
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#L44,59,97,122,148,169,173,181,186,191,213,214,215,216,217,218,219,220,232
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 USDT
uint256 public UsdtHardCap
uint256 _amount
address _referrer
address _user
uint256 _minAmount
uint256 _maxAmount
uint256 _time
address payable _newOwner
uint256 _value
uint256 _airDropRefPercentage
uint256 _percentageDivider
uint256 _price
uint256 _soldToken

...
```

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#L187
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#L170,182,222
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.

```
airDropAmount = _amount * 10**token.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#L149,152
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**token.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#L187
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#L5
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 ^0.8.14;
```

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#L115,142,143,159,161,192
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)
token.transfer(msg.sender, user.referrerReward)
token.transfer(msg.sender, user.claimAbleAmount)
IBEP20(token).transfer(msg.sender, airDropAmount)

IBEP20(token).transfer(
                user.referrer,
                (airDropAmount * airDropRefPercentage) /
percentageDivider
            )
USDT.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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	PreSale.sol#L248
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {}
```

Recommendation

The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

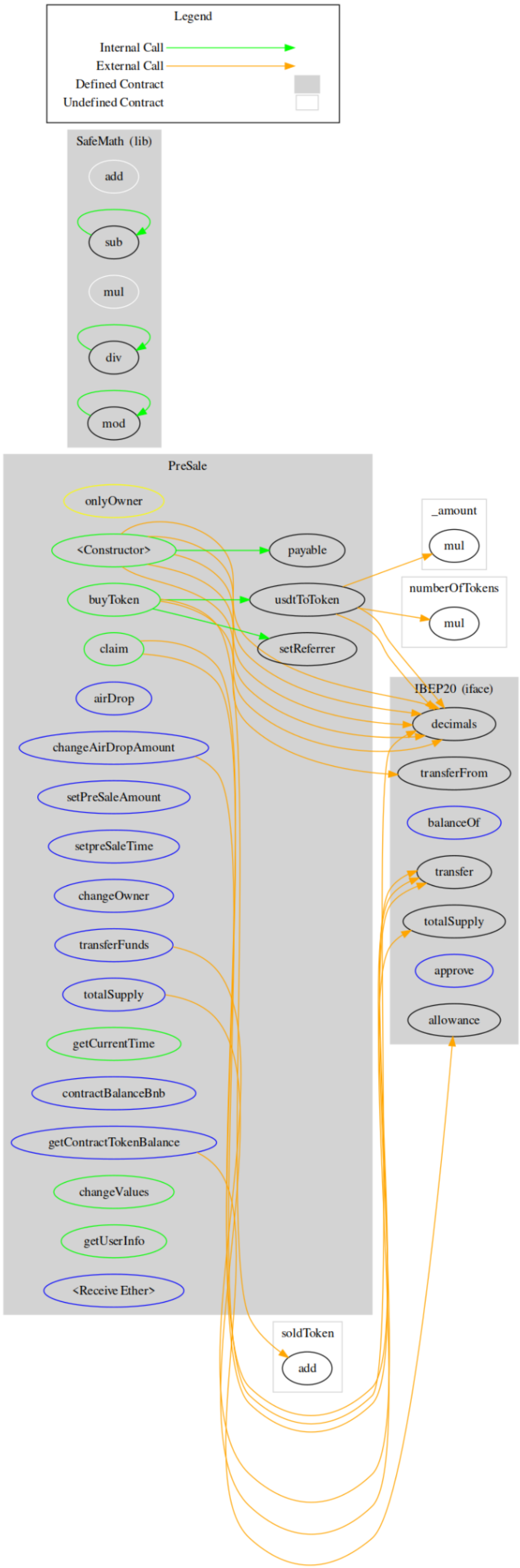
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
PreSale	Implementation			
		Public	✓	-
	buyToken	Public	✓	-
	setReferrer	Internal	✓	
	claim	Public	✓	-
	usdtToToken	Public		-
	airDrop	External	✓	-
	changeAirDropAmount	External	✓	onlyOwner
	setPreSaleAmount	External	✓	onlyOwner

	setpreSaleTime	External	✓	onlyOwner
	changeOwner	External	✓	onlyOwner
	transferFunds	External	✓	onlyOwner
	totalSupply	External		-
	getCurrentTime	Public		-
	contractBalanceBnb	External		-
	getContractTokenBalance	External		-
	changeValues	Public	✓	onlyOwner
	getUserInfo	Public		-
		External	Payable	-
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
<Constructor>
payable
buyToken
usdtToToken
claim
setReferrer
airDrop
changeAirDropAmount
setPreSaleAmount
setpreSaleTime
changeOwner
transferFunds
totalSupply
getCurrentTime
contractBalanceBnb
getContractTokenBalance
changeValues
getUserInfo
<Receive Ether>
_amount
mul
numberOfTokens
mul
IBEP20 (iface)
decimals
transferFrom
balanceOf
transfer
totalSupply
approve
allowance
soldToken
add

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