Cyberscope

Audit Report

AI MASA

July 2023

Network	BSC
Address	0xB6c56e4b12a4c3ACDc7E9e2F393e2dEc334b10a0
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PFM	Potential Functions Misuse	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **4**
Audit Updates 4
Source Files 4
Findings Breakdown **7**
PFM - Potential Functions Misuse 8
Description 8
Recommendation 8
L19 - Stable Compiler Version 9
Description 9
Recommendation 9
Functions Analysis **10**
Inheritance Graph **11**
Flow Graph **12**
Summary **13**
Corrected Phase 3, 3 Aug 2023 13
Disclaimer **14**
About Cyberscope **15**

Review

Explorer	https://bscscan.com/address/0xb6c56e4b12a4c3acdc7e9e2f393e2dec334b10a0

Audit Updates

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

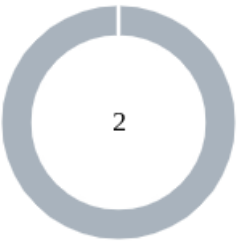
Source Files

Filename	SHA256
contracts/test.sol	546040540436aed94c2c9f0bc255068c607f030c8fd4277629c6e8011df8a4df
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	357c8d1a0fb673fa10a884d6e27b383171ecc3eaf8dee8211de75f88ff77843d
@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol	5b478023a1200e1364308ca06cdefec7cb7ab990a1cb904cbbdbaa7ba85076be
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	5c1ac829a429b0c2ca9b4c9ed8b78d412320e9175e45f088c4e9056ef95fbf21
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4

@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	db92fc1b515decad3a783b1422190877d2d70b907c6e36fb0998d9465aee42db
@openzeppelin/contracts-upgradeable/utils/math/SignedMathUpgradeable.sol	4f06981f993ea4a96e078c2036b5a42e1edade38996e5180171d5fe4be2f18fe
@openzeppelin/contracts-upgradeable/utils/math/SafeCastUpgradeable.sol	647d03e70d45c15cd9aa3afc3b32de945ec024a022614e263f33bb35c557ac94
@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol	fbf7ebc0f3c2cf5aef908ecce85e69af53db4e2c6652f61c8ac1e3f416c2fa99
@openzeppelin/contracts-upgradeable/utils/cryptography/EIP712Upgradeable.sol	ed30d96d25a360d320a807157db07b5bbb73392745ce8188b775787eb2d33fb9
@openzeppelin/contracts-upgradeable/utils/cryptography/ECDSAUpgradeable.sol	aefb3039d0aae994ad64c397dfa0bcc1ab6e675e2fd97b2fdfc0e7739def0b5b
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	78a6bc84bbb417f0d8a6b12e181e0f783151774f4f0c054c5d3f920e70d69f8c
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	9619cf23b549a5126042a4e20b09a2eb12dc8c2975258e3b8cde79cc593b6926
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-ERC20PermitUpgradeable.sol	a08be4078da127929eb0d760949376defa730f6af97f022822399b0ad880ad03
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20PermitUpgradeable.sol	cf0f8a5ee1c560ad1c5b0847a1531c5904cee45d0ec811cd83d0d95cbd5a333b
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol	68bcca423fc72ec9625e219c9e36306c726a347e43f3711467c579bd3f6500c8
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20VotesUpgradeable.sol	b89e9ae4dcbdd5b05db7ef2c7ee993f416d03b0787e613c33324dc480a65bbc0
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20PermitUpgradeable.sol	dbd6de9e5c4479ed83e3106f5e1d03ed91a4d37e97b24e65e345366ec879d979

@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol	ca660e828b0c4be205a9f56f3b87b91c1fa67cfd0f6e9dbd431faea7a6280d36
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	04338003a3be8f5f38595048b591d80fdc147bf95cc7c6285e1e1a5f1afa2b47
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a2c4e5c274a586f145d278293ae33198cd8f412ab7e6d26f2394c8949b32b24b
@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol	e0ac7115916f0dce0a8e80769694736f3e674bdc5b2e5853964c82004b1e1cc5
@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol	40dd5b14a370eea51ba94eb1b66a89638c6c54d86cc9f406599075c273e5e4c6
@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol	a94576fd98585c07b2a9725f7c89c910a3a1909a03f49ec2df465327c6a0ffc3
@openzeppelin/contracts-upgradeable/interfaces/IERC6372Upgradeable.sol	a651c2fe286001386424f9ee592ffe45d8675d3512cce47e4274b587f4794772
@openzeppelin/contracts-upgradeable/interfaces/IERC5805Upgradeable.sol	ae6f56560f3313a609ab2878ead6ec287d27615c4d258194c244cceeeedd3ee3
@openzeppelin/contracts-upgradeable/interfaces/IERC5267Upgradeable.sol	6a0d92d0222dd70cdc073029b6fe979e03e65d9c9ea4f2b8ffb774e144d2a51e
@openzeppelin/contracts-upgradeable/interfaces/IERC1967Upgradeable.sol	167828e6f725b1d47d82bc912fd0f1c6ed0fb67a4e5e06a4d62e72b4a53e95cf
@openzeppelin/contracts-upgradeable/governance/utils/IVotesUpgradeable.sol	f1546747e3834205ca3358625f8a8e1de2e17912b94d0c3c9703a6a57e93d0b4
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	1fbf2a131b895514f0027866cc0deff151ea16424b4aed2b8c573d2275cfa9e8

Findings Breakdown



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

PFM - Potential Functions Misuse

Criticality	Minor / Informative
Location	contracts/UUPSUpgradeable.sol#L74,89
Status	Unresolved

Description

The contract contains the `upgradeTo` and `upgradeToAndCall` functions that facilitate the upgradeability of the proxy's implementation. While these functions are essential for maintaining and improving the contract over time, there exists a potential risk. If these functions misused, they could redirect the proxy to point to a malicious or unintended implementation. Such a scenario could compromise the contract's intended behavior, potentially leading to loss of funds, unauthorized access, or unintended fuctionalities.

```
    function upgradeTo(address newImplementation) external
virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, new bytes(0),
false);
    }

    function upgradeToAndCall(address newImplementation, bytes
memory data) external payable virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, data, true);
    }
```

Recommendation

It is recommended to implement robust access controls and governance mechanisms around the `upgradeTo` and `upgradeToAndCall` functions. Only trusted entities, such as contract administrators or a multi-signature wallet, should have the authority to invoke these functions.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/token1.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 ^0.8.9;
```

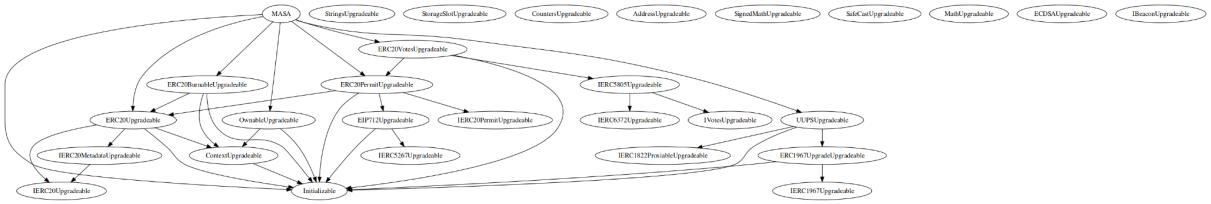
Recommendation

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.

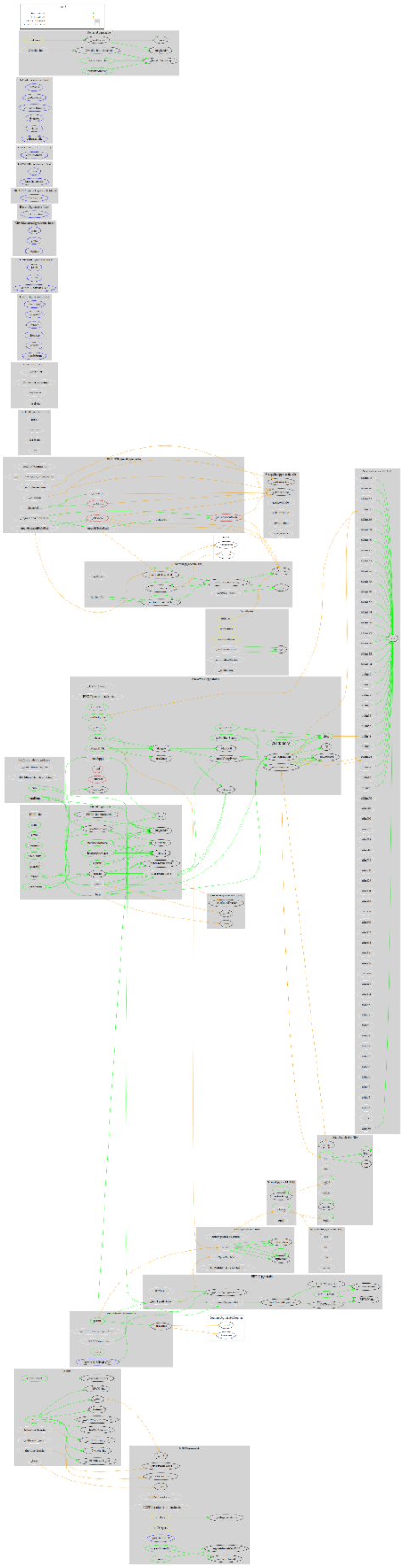
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MASA	Implementation	Initializable, ERC20Upgradeable, ERC20BurnableUpgradeable, OwnableUpgradeable, ERC20PermitUpgradeable, ERC20VotesUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	_beforeTokenTransfer	Internal	✓	
	_authorizeUpgrade	Internal	✓	onlyOwner
	_afterTokenTransfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

Inheritance Graph



Flow Graph



Summary

AI MASA contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. AI MASA is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

Corrected Phase 3, 3 Aug 2023

At the time of the audit report, the contract with address 0xB6c56e4b12a4c3ACDc7E9e2F393e2dEc334b10a0 is pointed by the following proxy address: 0x6730C6B035cD9bF185cd86d74204F30b3F260130.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

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

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



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

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

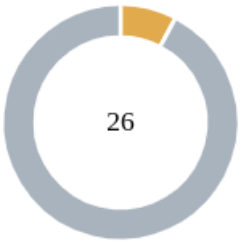
Overview of Functionality:

- 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
Medium	UC	Unreachable Code	Unresolved
Medium	US	Untrusted Source	Unresolved
Minor / Informative	OCTD	Transfers Contract's Tokens	Unresolved
Minor / Informative	RCC	Redundant Condition Check	Unresolved
Minor / Informative	DKO	Delete Keyword Optimization	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	MC	Missing Check	Unresolved
Minor / Informative	PBV	Percentage Boundaries Validation	Unresolved
Minor / Informative	PVI	Presale Variables Inconsistency	Unresolved
Minor / Informative	PII	Presale Initialization Inconsistency	Unresolved
Minor / Informative	RC	Redundant Calculations	Unresolved
Minor / Informative	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.

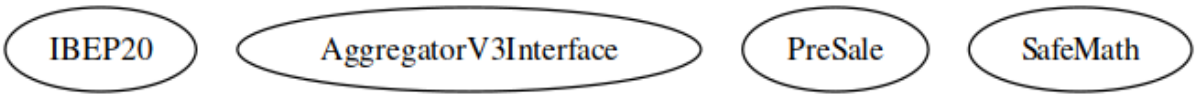
Functions Analysis

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

Cyberscope

Audit Report

AI MASA

July 2023

Network	BSC Testnet
Address	0x8073D97B896299188681999E05DdF046fEF73fc8
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PFM	Potential Functions Misuse	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **4**
Audit Updates 4
Source Files 4
Findings Breakdown **7**
ST - Stops Transactions 8
Description 8
Recommendation 8
MT - Mints Tokens 9
Description 9
Recommendation 9
PFM - Potential Functions Misuse 10
Description 10
Recommendation 10
L19 - Stable Compiler Version 11
Description 11
Recommendation 11
Functions Analysis **12**
Inheritance Graph **13**
Flow Graph **14**
Summary **15**
Initial Audit, 19 Jul 2023 15
Disclaimer **16**
About Cyberscope **17**

Review

Contract Name	MASA
Compiler Version	v0.8.16+commit.07a7930e
Optimization	800 runs
Explorer	https://testnet.bscscan.com/address/0x8073d97b896299188681999e05ddf046fef73fc8
Address	0x8073d97b896299188681999e05ddf046fef73fc8
Network	BSC_TESTNET
Decimals	18

Audit Updates

Initial Audit	19 Jul 2023

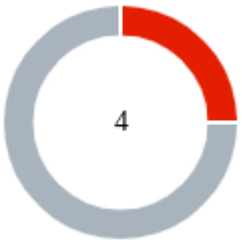
Source Files

Filename	SHA256
contracts/MASA.sol	84a78f52b4ae4382120f93ca3e65628d4ce62fa1f64efcfa33fb898e267fa335
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	68f5690fc266a6b48386c28cbfd72ec67c24b05a51ce26d24103577c15f61401
@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol	05b696b46ca1be28e19dfba65ea71c3b3615bd39d19bfd8212864a16c54870fd

@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	5c1ac829a429b0c2ca9b4c9ed8b78d412320e9175e45f088c4e9056ef95fbf21
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	1d7d481b79fd54d957c9a0696f6227f7799fec01d8ba41f5c130a7cc6b4eddc9
@openzeppelin/contracts-upgradeable/utils/math/SafeCastUpgradeable.sol	647d03e70d45c15cd9aa3afc3b32de945ec024a022614e263f33bb35c557ac94
@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol	158a0316fa289fad12c2ca764449e43e6724fb79c58fc438508d116f9af46b39
@openzeppelin/contracts-upgradeable/utils/cryptography/EIP712Upgradeable.sol	91e9d20515fa1516a9e9dd754b8a3ced5552f955b039dbe69d08c566fbd2e024
@openzeppelin/contracts-upgradeable/utils/cryptography/ECDSAUpgradeable.sol	2aee2a508bebf8e55bf78814d9d66a7a21c35c171e4010dfc3888c031f193628
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	4e09a7479aa3e7c313f8fc141c4c8fc04e0abfeb8754615ef7d78ec94c298b07
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	7307fb68607d3c93995797209010e5048c9cc1777f3b97dc7940f41a7327d080
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol	b97515a88e75c313eacf0a27c9439ef371d86d4c2730d3b13076640942f813df
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-ERC20PermitUpgradeable.sol	6d6ffe69a38a39c69acde1dd5edb74f80cff046c4a66d1cd816b98ca741c9a43
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol	68bcca423fc72ec9625e219c9e36306c726a347e43f3711467c579bd3f6500c8
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20VotesUpgradeable.sol	88763a9a0b498ca738c9a1c0c33a56464e0e8a2ad466426fe10b01cd9e01e2ae

@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol	ca660e828b0c4be205a9f56f3b87b91c1fa67cfd0f6e9dbd431faea7a6280d36
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	c05b019a0b3bee8f3fac2da7c929f7d665b97d6d046aa35126615fff11205119
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	7517b26ac0cee066447b94cbf7df8ad5ce91cc6ddf0fd1e3425fe978889f5eb0
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	98ce2984e449716f24043a8c11bbe969a6d34878b1d522b92c88d62708ba3376
@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol	e0ac7115916f0dce0a8e80769694736f3e674bdc5b2e5853964c82004b1e1cc5
@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol	f6c1a8b4512e9cc0168278c2a634b184fd86b1e39c7c283bcf34fb154236fc5d
@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol	a94576fd98585c07b2a9725f7c89c910a3a1909a03f49ec2df465327c6a0ffc3
@openzeppelin/contracts-upgradeable/governance/utils/IVotesUpgradeable.sol	400936c02700eb4147c65a91a15fb6f90d074d7519f8ebce49dce78a2c917186
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	da66c17044345dc892d85bd7ddc9745d25df0b3dacfba8f84eb87c60d6e40fe3

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	3

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	0

ST - Stops Transactions

Criticality	Minor / Informative
Location	contracts/MASA.sol#L39
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users including the owner, by calling the `pause` method.

```
    function pause() public onlyOwner {
        _pause();
    }
```

Recommendation

It is recommended to consider removing the `pause` functionality from the contract entirely, especially if there are no other intended purposes or justifiable use cases for it. Also 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.

MT - Mints Tokens

Criticality	Critical
Location	contracts/MASA.sol#L47
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address to, uint256 amount) public onlyOwner {
      _mint(to, amount);
  }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

PFM - Potential Functions Misuse

Criticality	Minor / Informative
Location	contracts/UUPSUpgradeable.sol#L72,85
Status	Unresolved

Description

The contract contains the `upgradeTo` and `upgradeToAndCall` functions that facilitate the upgradeability of the proxy's implementation. While these functions are essential for maintaining and improving the contract over time, there exists a potential risk. If these functions misused, they could redirect the proxy to point to a malicious or unintended implementation. Such a scenario could compromise the contract's intended behavior, potentially leading to loss of funds, unauthorized access, or unintended fuctionalities.

```
   function upgradeTo(address newImplementation) external
virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, new bytes(0),
false);
    }

    function upgradeToAndCall(address newImplementation, bytes
memory data) external payable virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, data, true);
    }
```

Recommendation

It is recommended to implement robust access controls and governance mechanisms around the `upgradeTo` and `upgradeToAndCall` functions. Only trusted entities, such as contract administrators or a multi-signature wallet, should have the authority to invoke these functions.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/MASA.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 ^0.8.9;
```

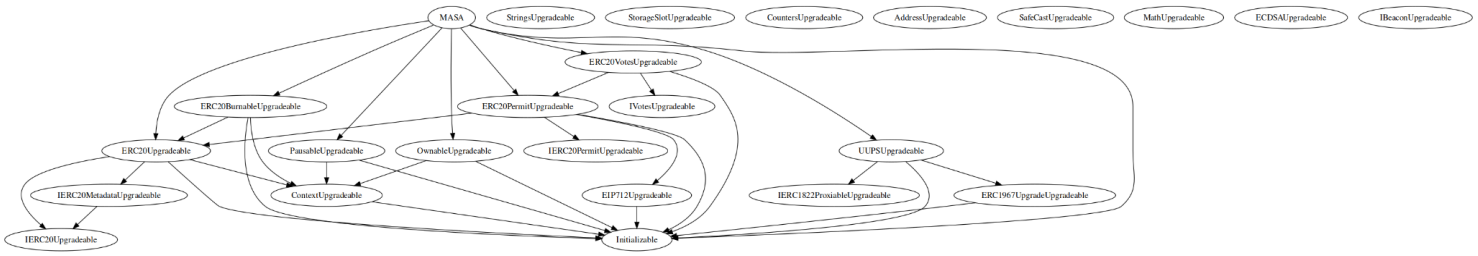
Recommendation

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.

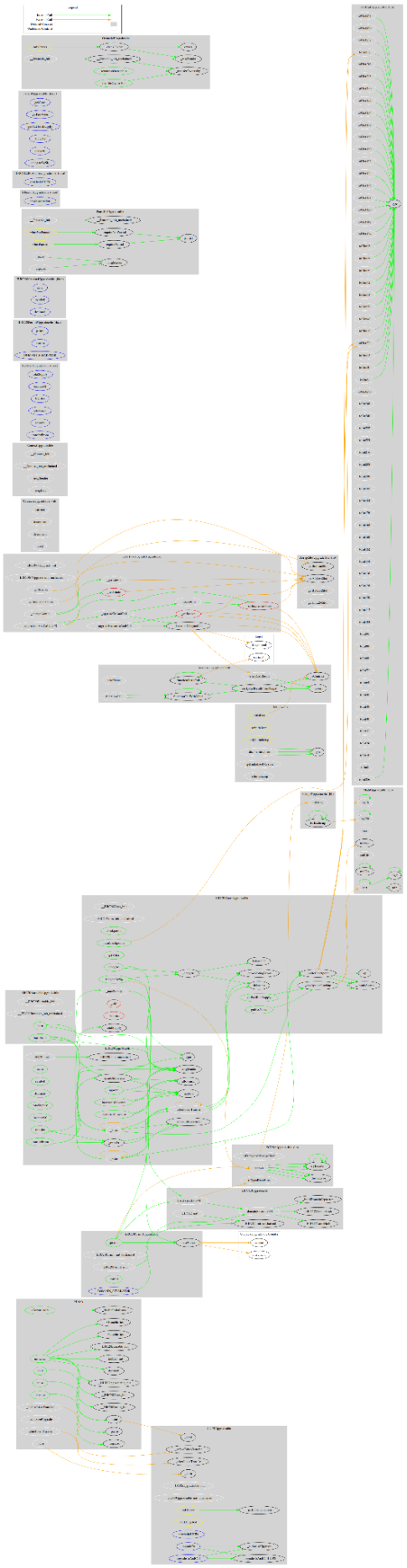
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MASA	Implementation	Initializable, ERC20Upgradeable, ERC20BurnableUpgradeable, PausableUpgradeable, OwnableUpgradeable, ERC20PermitUpgradeable, ERC20VotesUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	mint	Public	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	whenNotPaused
	_authorizeUpgrade	Internal	✓	onlyOwner
	_afterTokenTransfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

Inheritance Graph



Flow Graph



Summary

AI MASA contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and mint tokens. If the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Initial Audit, 19 Jul 2023

At the time of the audit report, the contract with address

0x8073D97B8962991886819 99E05DdF046fEF73fc8 is pointed by the following proxy

address: 0x21BAEAE318F0B2B4D6FaAEc38bd3E5572905aAC6.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

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

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



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

AI MASA

May 2023

Network	BSC Testnet
Address	0xbd4cfC9B4A3AC2947A51eA600daF13afafF41579
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **4**
Audit Updates 4
Source Files 4
Overview **5**
Functionality 5
Findings Breakdown **6**
Diagnostics **7**
IVA - Inconsistent Variable Amounts 9
Description 9
Recommendation 10
RC - Redundant Calculations 11
Description 11
Recommendation 11
PII - Presale Initialization Inconsistency 12
Description 12
Recommendation 12
ZD - Zero Division 14
Description 14
Recommendation 15
PVI - Presale Variables Inconsistency 16
Description 16
Recommendation 17
IUA - Incorrect USDT Address 18
Description 18
Recommendation 18
MEM - Misleading Error Messages 19
Description 19
Recommendation 19
PBV - Percentage Boundaries Validation 20
Description 20
Recommendation 20
MC - Missing Check 21
Description 21
Recommendation 21
DKO - Delete Keyword Optimization 22
Description 22
Recommendation 22
PTAI - Potential Transfer Amount Inconsistency 23

Description 23
Recommendation 23
MEE - Missing Events Emission 25
Description 25
Recommendation 26
AOI - Arithmetic Operations Inconsistency 27
Description 27
Recommendation 27
RSML - Redundant SafeMath Library 28
Description 28
Recommendation 28
RSK - Redundant Storage Keyword 29
Description 29
Recommendation 29
L04 - Conformance to Solidity Naming Conventions 30
Description 30
Recommendation 31
L06 - Missing Events Access Control 32
Description 32
Recommendation 32
L07 - Missing Events Arithmetic 33
Description 33
Recommendation 33
L13 - Divide before Multiply Operation 34
Description 34
Recommendation 34
L16 - Validate Variable Setters 35
Description 35
Recommendation 35
L19 - Stable Compiler Version 36
Description 36
Recommendation 36
L20 - Succeeded Transfer Check 37
Description 37
Recommendation 37
L22 - Potential Locked Ether 38
Description 38
Recommendation 38
Functions Analysis **39**
Inheritance Graph **41**
Flow Graph **42**
Summary **43**

Disclaimer **44**
About Cyberscope **45**

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	83c93798a46e2e2233a6b7773f3624e502dc61ae56da15054a7e71151059b746

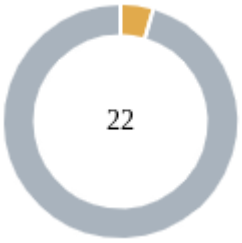
Overview

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



Critical	0
Medium	1
Minor / Informative	21

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
Medium	IVA	Inconsistent Variable Amounts	Unresolved
Minor / Informative	RC	Redundant Calculations	Unresolved
Minor / Informative	PII	Presale Initialization Inconsistency	Unresolved
Minor / Informative	ZD	Zero Division	Unresolved
Minor / Informative	PVI	Presale Variables Inconsistency	Unresolved
Minor / Informative	IUA	Incorrect USDT Address	Unresolved
Minor / Informative	PBV	Percentage Boundaries Validation	Unresolved
Minor / Informative	MC	Missing Check	Unresolved
Minor / Informative	DKO	Delete Keyword Optimization	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	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

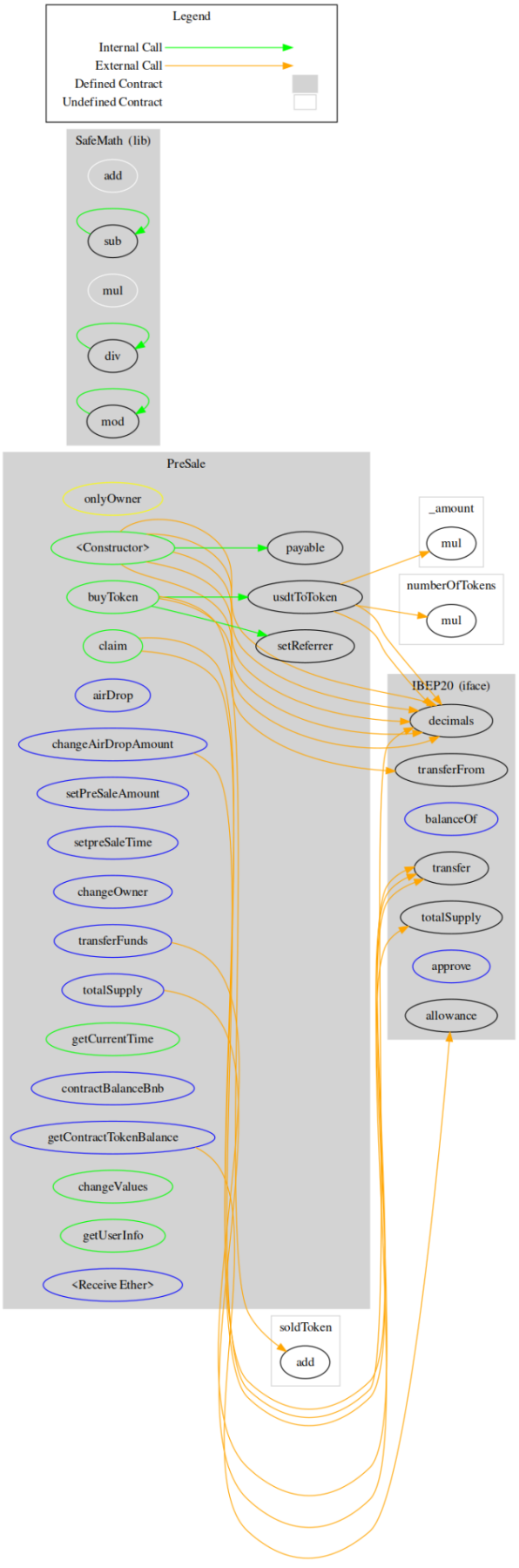
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
PreSale	Implementation			
		Public	✓	-
	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
setReferrer
claim
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

Cyberscope

Audit Report

AI MASA

July 2023

Network	BSC Testnet
Address	0x68A8152720df8724617c68Bfe913cAC509335fea
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PFM	Potential Functions Misuse	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **4**
Audit Updates 4
Source Files 4
Findings Breakdown **8**
MT - Mints Tokens 9
Description 9
Recommendation 9
PFM - Potential Functions Misuse 10
Description 10
Recommendation 10
L19 - Stable Compiler Version 11
Description 11
Recommendation 11
Functions Analysis **12**
Inheritance Graph **13**
Flow Graph **14**
Summary **15**
Disclaimer **16**
About Cyberscope **17**

Review

Explorer	https://testnet.bscscan.com/address/0x68a8152720df8724617c68bfe913cac509335fea

Audit Updates

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

Source Files

Filename	SHA256
contracts/token1.sol	b86a53023aae7ca8cc8826d4f21f50b66356fce5b15801aaf65db289b74ba287
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	357c8d1a0fb673fa10a884d6e27b383171ecc3eaf8dee8211de75f88ff77843d
@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol	5b478023a1200e1364308ca06cdefec7cb7ab990a1cb904cbbdbaa7ba85076be
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	5c1ac829a429b0c2ca9b4c9ed8b78d412320e9175e45f088c4e9056ef95fbf21

@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	db92fc1b515decad3a783b1422190877d2d70b907c6e36fb0998d9465aee42db
@openzeppelin/contracts-upgradeable/utils/math/SignedMathUpgradeable.sol	4f06981f993ea4a96e078c2036b5a42e1edade38996e5180171d5fe4be2f18fe
@openzeppelin/contracts-upgradeable/utils/math/SafeCastUpgradeable.sol	647d03e70d45c15cd9aa3afc3b32de945ec024a022614e263f33bb35c557ac94
@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol	fbf7ebc0f3c2cf5aef908ecce85e69af53db4e2c6652f61c8ac1e3f416c2fa99
@openzeppelin/contracts-upgradeable/utils/cryptography/EIP712Upgradeable.sol	ed30d96d25a360d320a807157db07b5bbb73392745ce8188b775787eb2d33fb9
@openzeppelin/contracts-upgradeable/utils/cryptography/ECDSAUpgradeable.sol	aefb3039d0aae994ad64c397dfa0bcc1ab6e675e2fd97b2fdfc0e7739def0b5b
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	78a6bc84bbb417f0d8a6b12e181e0f783151774f4f0c054c5d3f920e70d69f8c
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	9619cf23b549a5126042a4e20b09a2eb12dc8c2975258e3b8cde79cc593b6926
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-ERC20PermitUpgradeable.sol	a08be4078da127929eb0d760949376defa730f6af97f022822399b0ad880ad03
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20PermitUpgradeable.sol	cf0f8a5ee1c560ad1c5b0847a1531c5904cee45d0ec811cd83d0d95cbd5a333b

@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol	68bcca423fc72ec9625e219c9e36306c726a347e43f3711467c579bd3f6500c8
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20VotesUpgradeable.sol	b89e9ae4dcbdd5b05db7ef2c7ee993f416d03b0787e613c33324dc480a65bbc0
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20PermitUpgradeable.sol	dbd6de9e5c4479ed83e3106f5e1d03ed91a4d37e97b24e65e345366ec879d979
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol	ca660e828b0c4be205a9f56f3b87b91c1fa67cfd0f6e9dbd431faea7a6280d36
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	04338003a3be8f5f38595048b591d80fdc147bf95cc7c6285e1e1a5f1afa2b47
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a2c4e5c274a586f145d278293ae33198cd8f412ab7e6d26f2394c8949b32b24b
@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol	e0ac7115916f0dce0a8e80769694736f3e674bdc5b2e5853964c82004b1e1cc5
@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol	40dd5b14a370eea51ba94eb1b66a89638c6c54d86cc9f406599075c273e5e4c6
@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol	a94576fd98585c07b2a9725f7c89c910a3a1909a03f49ec2df465327c6a0ffc3
@openzeppelin/contracts-upgradeable/interfaces/IERC6372Upgradeable.sol	a651c2fe286001386424f9ee592ffe45d8675d3512cce47e4274b587f4794772
@openzeppelin/contracts-upgradeable/interfaces/IERC5805Upgradeable.sol	ae6f56560f3313a609ab2878ead6ec287d27615c4d258194c244cceeeedd3ee3

@openzeppelin/contracts-upgradeable/interfaces/IERC5267Upgradeable.sol	6a0d92d0222dd70cdc073029b6fe979e03e65d9c9ea4f2b8ffb774e144d2a51e
@openzeppelin/contracts-upgradeable/interfaces/IERC1967Upgradeable.sol	167828e6f725b1d47d82bc912fd0f1c6ed0fb67a4e5e06a4d62e72b4a53e95cf
@openzeppelin/contracts-upgradeable/governance/utils/IVotesUpgradeable.sol	f1546747e3834205ca3358625f8a8e1de2e17912b94d0c3c9703a6a57e93d0b4
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	1fbf2a131b895514f0027866cc0deff151ea16424b4aed2b8c573d2275cfa9e8

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	2	0	0	0

MT - Mints Tokens

Criticality	Critical
Location	contracts/token1.sol#L36
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address to, uint256 amount) public onlyOwner {
      _mint(to, amount);
  }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

PFM - Potential Functions Misuse

Criticality	Minor / Informative
Location	contracts/UUPSUpgradeable.sol#L74,89
Status	Unresolved

Description

The contract contains the `upgradeTo` and `upgradeToAndCall` functions that facilitate the upgradeability of the proxy's implementation. While these functions are essential for maintaining and improving the contract over time, there exists a potential risk. If these functions misused, they could redirect the proxy to point to a malicious or unintended implementation. Such a scenario could compromise the contract's intended behavior, potentially leading to loss of funds, unauthorized access, or unintended fuctionalities.

```
    function upgradeTo(address newImplementation) external
virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, new bytes(0),
false);
    }

    function upgradeToAndCall(address newImplementation, bytes
memory data) external payable virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, data, true);
    }
```

Recommendation

It is recommended to implement robust access controls and governance mechanisms around the `upgradeTo` and `upgradeToAndCall` functions. Only trusted entities, such as contract administrators or a multi-signature wallet, should have the authority to invoke these functions.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/token1.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 ^0.8.9;
```

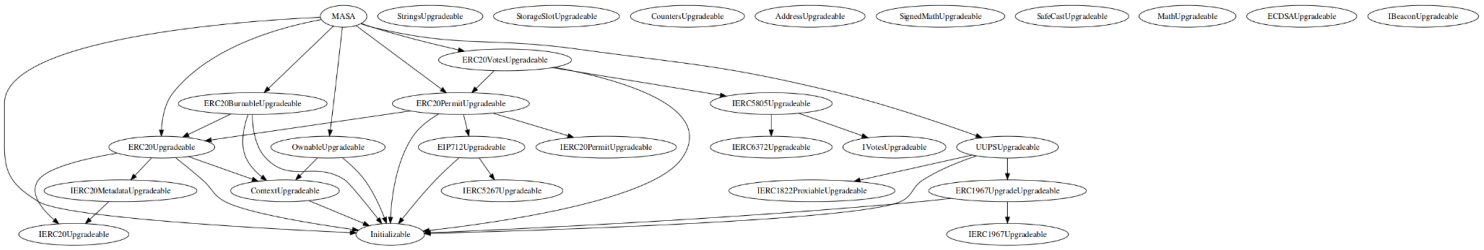
Recommendation

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.

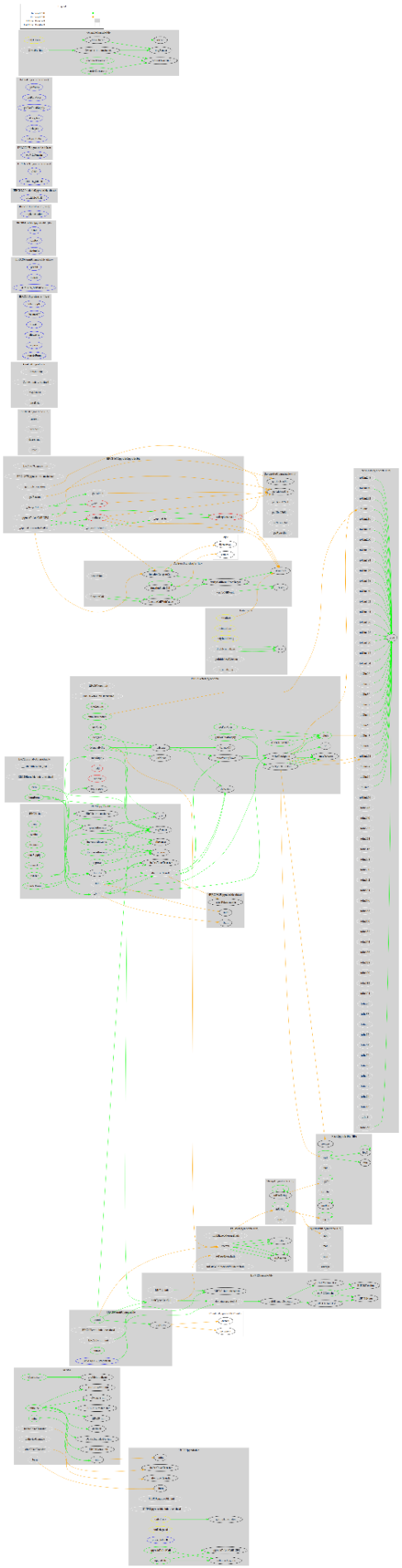
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MASA	Implementation	Initializable, ERC20Upgradeable, ERC20BurnableUpgradeable, OwnableUpgradeable, ERC20PermitUpgradeable, ERC20VotesUpgradeable, UUPSUpgradeable		
		Public	✓	-
	initialize	Public	✓	initializer
	mint	Public	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	
	_authorizeUpgrade	Internal	✓	onlyOwner
	_afterTokenTransfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

Inheritance Graph



Flow Graph



Summary

AI MASA contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like mint tokens. If the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

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

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



The Cyberscope team

https://www.cyberscope.io

Cyberscope

Audit Report

AI MASA

July 2023

Network BSC Testnet

Address 0x47159a9CBfe56Bd7fEB18022Ef091DBCE65807C6

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **4**
Audit Updates 4
Source Files 4
Overview **5**
Functionality 5
Findings Breakdown **7**
Diagnostics **8**
SUF - Stuck USDT Funds 10
Description 10
Recommendation 10
UC - Unreachable Code 11
Description 11
Recommendation 11
RPC - Redundant Precision Calculation 13
Description 13
Recommendation 13
RCF - Reusable Code Functionality 14
Description 14
Recommendation 14
DPI - Decimals Precision Inconsistency 15
Description 15
Recommendation 16
CR - Code Repetition 17
Description 17
Recommendation 18
DKO - Delete Keyword Optimization 19
Description 19
Recommendation 19
PTAI - Potential Transfer Amount Inconsistency 20
Description 20
Recommendation 20
MEE - Missing Events Emission 22
Description 22
Recommendation 23
ZD - Zero Division 24
Description 24
Recommendation 25
AOI - Arithmetic Operations Inconsistency 26

Description 26
Recommendation 26
MC - Missing Check 27
Description 27
Recommendation 27
PBV - Percentage Boundaries Validation 28
Description 28
Recommendation 28
PVI - Presale Variables Inconsistency 29
Description 29
Recommendation 30
IUA - Incorrect USDT Address 31
Description 31
Recommendation 31
PII - Presale Initialization Inconsistency 32
Description 32
Recommendation 32
RC - Redundant Calculations 34
Description 34
Recommendation 34
RSML - Redundant SafeMath Library 35
Description 35
Recommendation 35
RSK - Redundant Storage Keyword 36
Description 36
Recommendation 36
L04 - Conformance to Solidity Naming Conventions 37
Description 37
Recommendation 38
L06 - Missing Events Access Control 39
Description 39
Recommendation 39
L07 - Missing Events Arithmetic 40
Description 40
Recommendation 40
L13 - Divide before Multiply Operation 41
Description 41
Recommendation 41
L16 - Validate Variable Setters 42
Description 42
Recommendation 42
L19 - Stable Compiler Version 43

Description 43
Recommendation 43
L20 - Succeeded Transfer Check 44
Description 44
Recommendation 44
Functions Analysis **45**
Inheritance Graph **48**
Flow Graph **49**
Summary **50**
Disclaimer **51**
About Cyberscope **52**

Review

Contract Name	PreSale
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0x47159a9cbfe56bd7feb18022ef091dbce65807c6
Address	0x47159a9cbfe56bd7feb18022ef091dbce65807c6
Network	BSC_TESTNET

Audit Updates

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

Source Files

Filename	SHA256
PreSale.sol	06e53a183a3db65a2be2f236fa096ff3fee8a00a5d3439866f0c63f69bcc756a

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 `0x2514895c72f50D8bd4B4F9b1110F0D6bD2c97526`. 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

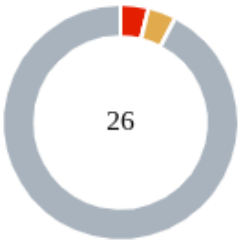
Overview of Functionality:

- 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	1
Medium	1
Minor / Informative	24

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	1	0	0	0
Minor / Informative	24	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	SUF	Stuck USDT Funds	Unresolved
Medium	UC	Unreachable Code	Unresolved
Minor / Informative	RPC	Redundant Precision Calculation	Unresolved
Minor / Informative	RCF	Reusable Code Functionality	Unresolved
Minor / Informative	DPI	Decimals Precision Inconsistency	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	DKO	Delete Keyword Optimization	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	ZD	Zero Division	Unresolved
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	MC	Missing Check	Unresolved
Minor / Informative	PBV	Percentage Boundaries Validation	Unresolved
Minor / Informative	PVI	Presale Variables Inconsistency	Unresolved

●	IUA	Incorrect USDT Address	Unresolved
●	PII	Presale Initialization Inconsistency	Unresolved
●	RC	Redundant Calculations	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

SUF - Stuck USDT Funds

Criticality	Critical
Location	PreSale.sol#L289
Status	Unresolved

Description

The contract is designed to accept USDT as payment through the `buyToken` function. However, while the contract can accumulate USDT, there is no mechanism to withdraw or utilize these USDT funds. The only withdrawal function, `transferFunds`, facilitates the transfer of the native token (in this case, BNB) from the contract to the owner. Consequently, any USDT sent to the contract remains trapped, as there is neither a withdrawal function for USDT nor a mechanism to convert USDT to BNB within the contract.

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

Recommendation

It is recommended to implement a function that allows the contract owner to withdraw USDT funds safely. This can be achieved by interacting with the USDT token contract's transfer or `transferFrom` functions. Additionally, to enhance flexibility, consider integrating a mechanism or external service that can convert USDT to BNB within the contract, ensuring that funds in any form can be easily accessed and managed. This approach will ensure that the contract remains functional, funds are not inadvertently locked, and the contract owner has full control over all assets within the contract.

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.

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.

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.

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.

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.

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#L207,208
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#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.

ZD - Zero Division

Criticality	Minor / Informative
Location	PreSale.sol#L230,256,323
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.

```
if (_val) {
    users[_referrer].referrerReward += usdtToToken(
    (_amount * referrerPercentage) / percentageDivider);
} else {
    users[_referrer].referrerReward += bnbToToken(_amount);
    ((_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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L139,151
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#L271,279
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#L213,248
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.

PVI - Presale Variables Inconsistency

Criticality	Minor / Informative
Location	PreSale.sol#L323
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 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 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(0x73445033cEA2d4b74c1c6119E1514da7B6a28739);
```

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.

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(0x73445033cEA2d4b74c1c6119E1514da7B6a28739);
        USDT = IBEP20(0x73445033cEA2d4b74c1c6119E1514da7B6a28739);
        priceFeedBnb = AggregatorV3Interface(
            0x2514895c72f50D8bd4B4F9b1110F0D6bD2c97526
        );
        referrerPercentage = 7_00;
        airDropRefPercentage = 5_00;
        percentageDivider = 100_00;
        airDropAmount = 100 * 10**token.decimals();
        tokenPerUsd = 1000;
        UsdtHardCap = 1000000000 * 10**USDT.decimals();
        tokenHardCap = 100000000000 * 10**token.decimals();
        minAmount = 100 * 10**token.decimals();
        maxAmount = 10000000 * 10**token.decimals();
        preSaleTime = block.timestamp + 3 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#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.

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#L315
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#L69,87,126,156,182,199,214,215,216,217,241,263,267,271,279,284,289,310,324,325,326,327,328,329,330,331,332
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
uint256 _value
address _user
bool _val
uint256 _tokenPerUsd
uint256 _minAmount
uint256 _maxAmount
uint256 _time
address payable _newOwner
uint256 _airDropRefPercentage
uint256 _percentageDivider

...
```

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#L285
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#L264,268,280,334
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**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#L184,187,242,245
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#L285
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#L148,207,208,252,254
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
            )
```

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.

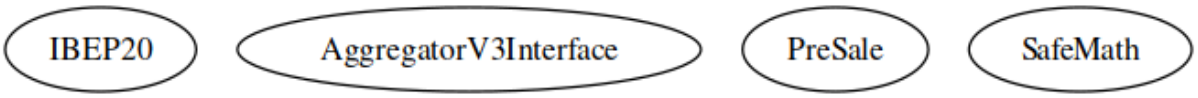
Functions Analysis

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
	totalSupply	External		-
	getCurrentTime	Public		-
	contractBalanceBnb	External		-
	getContractTokenBalance	External		-
	getUserInfo	Public		-
	changeValues	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
totalSupply
getCurrentTime
contractBalanceBnb
getContractTokenBalance
getUserInfo
changeValues
bnbToUsdt
setReferrer
payable
bnbToToken
usdtToToken
getLatestPriceBnb
AggregatorV3Interface (iface)
decimals
description
version
getRoundData
latestRoundData
precision
mul
bnbToUsd
mul
numberOfTokens
mul
_amount
mul
IBEP20 (iface)
transferFrom
decimals
balanceOf
transfer
totalSupply
approve
allowance
soldToken
add
owner
transfer

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