Cyberscope

Audit Report

INTDESTCOIN

May 2023

SHA256	8248d4543dc1dd1b57e7bf33141ac8bb98290862acf62c65ef013ced8f82373c
Audited by	© cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x057a1aa246ef1700621a7ffe4314be2f4d0470f2

Audit Updates

Initial Audit	17 May 2023
Corrected Phase 2	27 May 2023

Source Files

Filename	SHA256
contracts/testingDeploy/Presale.sol	8248d4543dc1dd1b57e7bf33141ac8bb98290862acf62c65ef013ced8f82373c

Introduction

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

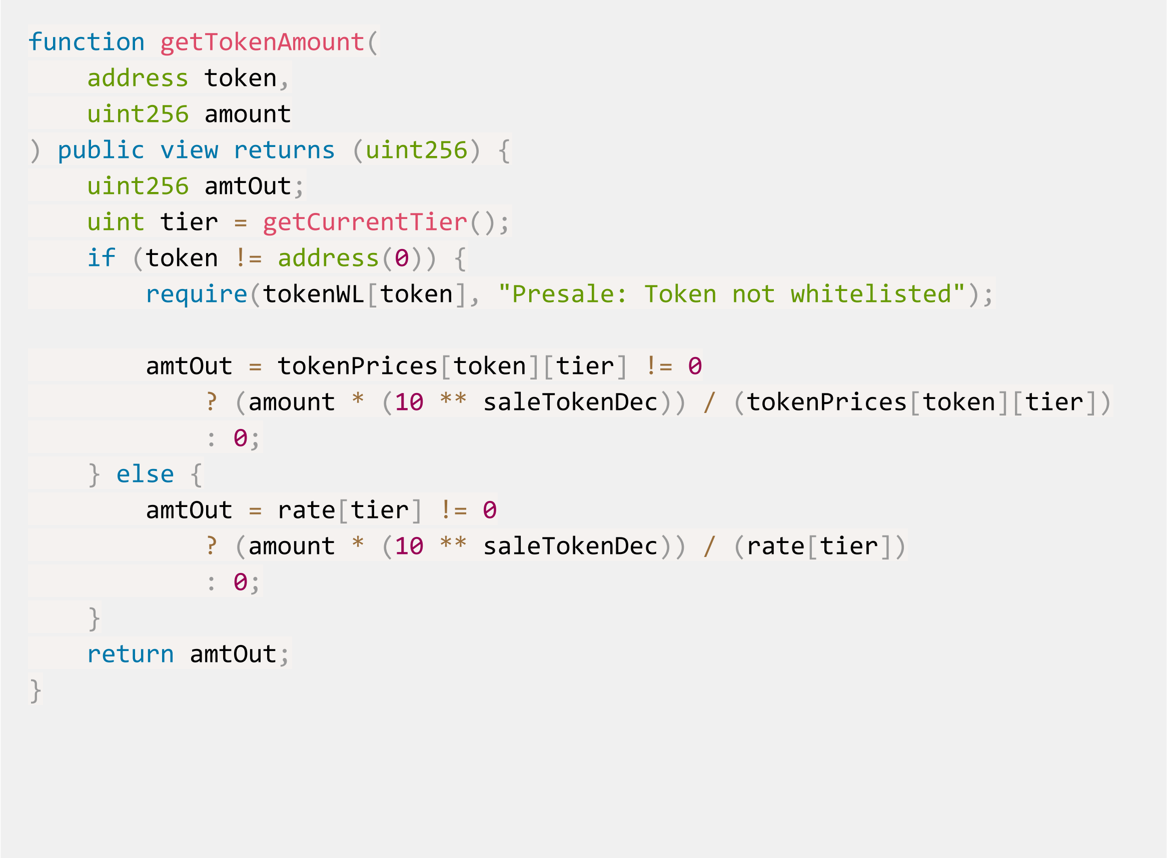
Functionality

Overview of Functionality:

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

Buy Mechanism Architecture

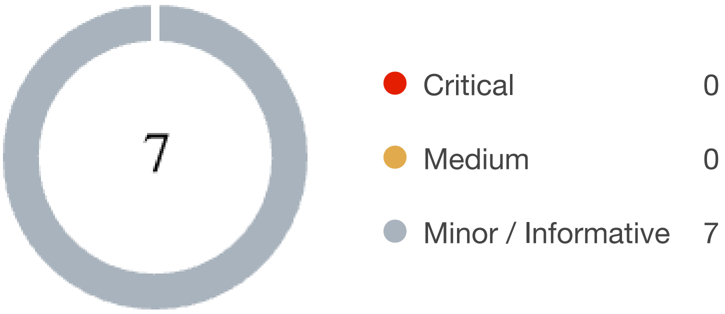
The presale contract is designed to support a selling rate that exceeds a 1:1 ratio between the presale token and the whitelisted tokens or the native currency. This means that it can be configured to sell presale tokens at a ratio of 2:1 or even higher. However, the contract can not be configured to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



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

        amtOut = tokenPrices[token][tier] != 0
            ? (amount * (10 ** saleTokenDec)) / (tokenPrices[token][tier])
            : 0;
    } else {
        amtOut = rate[tier] != 0
            ? (amount * (10 ** saleTokenDec)) / (rate[tier])
            : 0;
    }
    return amtOut;
}
```

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	7	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	BAPI	Bounce Amount Performance Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Presale.sol#L877
Status	Unresolved

Description

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

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

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

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

Recommendation

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

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

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

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	Presale.sol#L745
Status	Unresolved

Description

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

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

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

Recommendation

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

BAPI - Bounce Amount Performance Improvement

Criticality	Minor / Informative
Location	contracts/Presale.sol#L840
Status	Unresolved

Description

The contract iterates an array that is sorted ascending in order to match the proper percentage for the provided amount. The iteration algorithm always performs a full array iteration even if the amount has matched.

```
function getBounceAmount(uint256 amount) public view returns (uint256) {
    uint256 bounce = 0;
    for (uint256 i = 0; i < bounces.length; i++) {
        if (amount >= bounces[i].amount) {
            bounce = bounces[i].percentage;
        }
    }
    return (amount * bounce) / 1000;
}
```

Recommendation

The team is advised to exploit the sorted array structure and modify the for loop in order to early exit if the precondition has been fulfilled. A suggested implementation could be:

```
function getBounceAmount(uint256 amount) public view returns (uint256) {
    for (uint256 i = bounces.length ; i != 0; --i) {
        uint256 index = i - 1;
        if (amount >= bounces[index].amount) {
            return (amount * bounces[index].percentage) / 1000;
        }
    }

    return 0;
}
```

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/testingDeploy/Presale.sol#L691,692,718,719,732,733,741,746,747,766,767,852,853,903,907,912
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

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

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/testingDeploy/Presale.sol#L726,904,908
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.

```
presaleStartTime = _presaleStartTime
minBuyLimit = _minBuyLimit
maxBuyLimit = _maxBuyLimit
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/testingDeploy/Presale.sol#L264,296,340,358,375,393,477,495,511
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function functionCall(
        address target,
        bytes memory data
    ) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/testingDeploy/Presale.sol#L240,422
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            size := extcodesize(account)
        }

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

Recommendation

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

Functions Analysis

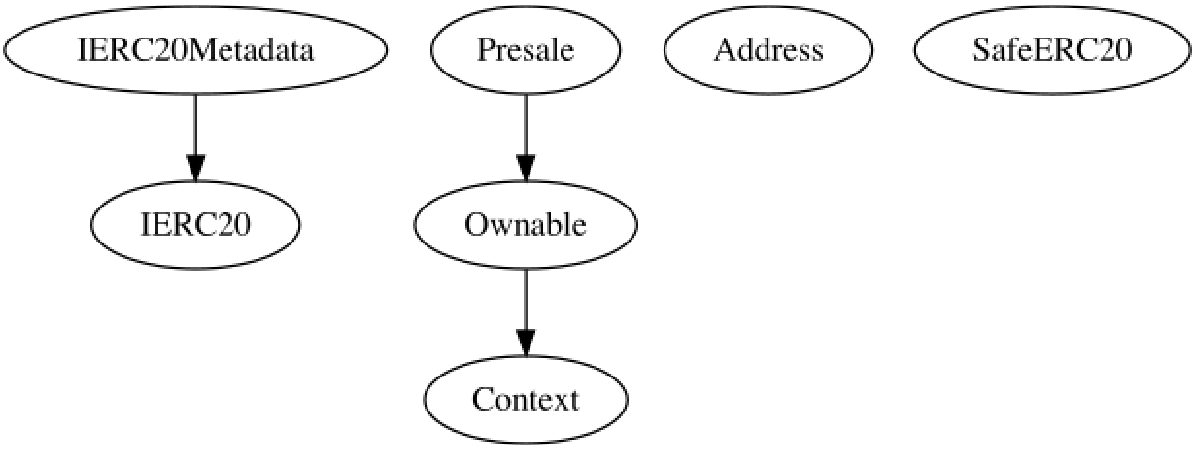
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

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

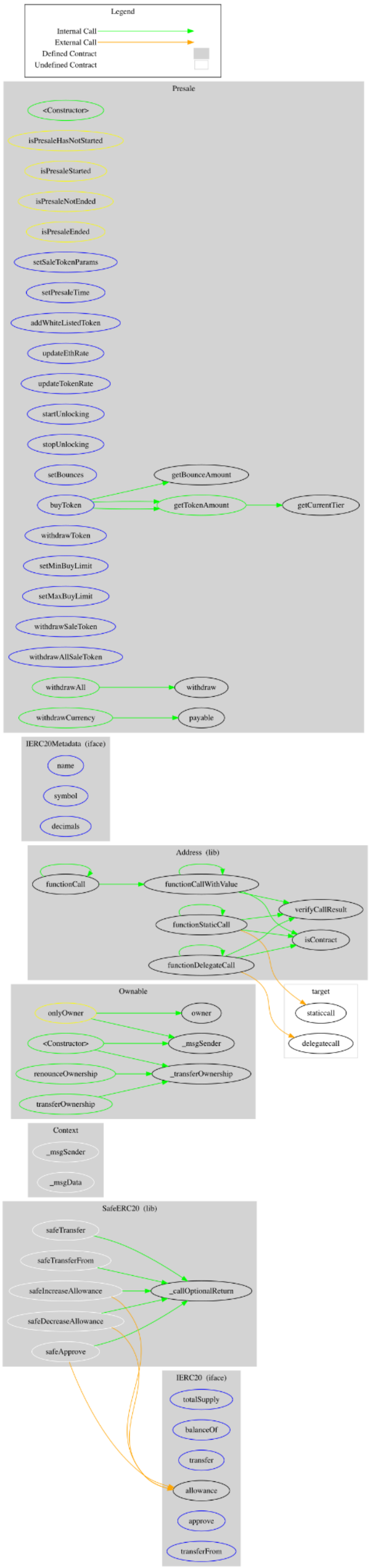
	decimals	External		-
Presale	Implementation	Ownable		
		Public	✓	-
	setSaleTokenParams	External	✓	onlyOwner isPresaleHasNotStarted
	setPresaleTime	External	✓	onlyOwner isPresaleHasNotStarted
	addWhiteListedToken	External	✓	onlyOwner
	updateEthRate	External	✓	onlyOwner
	updateTokenRate	External	✓	onlyOwner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

	withdrawAllSaleToken	External	✓	onlyOwner isPresaleEnded
	withdraw	Public	✓	onlyOwner
	withdrawAll	Public	✓	onlyOwner
	withdrawCurrency	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Presale
<Constructor>
isPresaleHasNotStarted
isPresaleStarted
isPresaleNotEnded
isPresaleEnded
setSaleTokenParams
setPresaleTime
addWhiteListedToken
updateEthRate
updateTokenRate
startUnlocking
stopUnlocking
setBounces
getBounceAmount
buyToken
getTokenAmount
getCurrentTier
withdrawToken
setMinBuyLimit
setMaxBuyLimit
withdrawSaleToken
withdrawAllSaleToken
withdrawAll
withdraw
withdrawCurrency
payable
IERC20Metadata (iface)
name
symbol
decimals
Address (lib)
functionCall
functionCallWithValue
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
Context
_msgSender
_msgData
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeIncreaseAllowance
_callOptionalReturn
safeDecreaseAllowance
safeApprove
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

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

Disclaimer

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