Cyberscope

Audit Report

Paragon Sale

June 2024

SHA256 45a74831080964b96766a0fbef95ce918d6d739d135f64e8dd8b08d1c1189f45

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x86d05e00ae88c0c220c8e25f078b154012c24c7b

Audit Updates

Initial Audit	07 Jun 2024

Source Files

Filename	SHA256
contracts/TokenSale.sol	45a74831080964b96766a0fbef95ce918d6d739d135f64e8dd8b08d1c1189f45
@openzeppelin/contracts/utils/Address.sol	b3710b1712637eb8c0df81912da3450da6ff67b0b3ed18146b033ed15b1aa3b9
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	471157c89111d7b9eab456b53ebe9042bc69504a64cb5cc980d38da9103379ae
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	912509e0e9bf74e0f8a8c92d031b5b26d2d35c6d4abf3f56251be1ea9ca946bf
@chainlink/contracts/src/v0.8/shared/interfaces/AggregatorV3Interface.sol	62d0c0c753b724d3450723960ca4d256a16613a8c50ca42755610a951b772c7d

Overview

The TokenSale contract is designed to facilitate the sale of tokens in exchange for Ether (ETH). It integrates the SafeERC20 library from OpenZeppelin to ensure safe and secure token transactions and utilizes Chainlink's AggregatorV3Interface for fetching the latest price of ETH in USD. The contract is initialized with the address of the token being sold and the address of a multisig wallet, which holds administrative privileges.

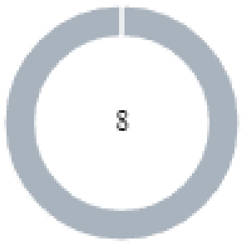
The contract includes several key functions to manage the token sale. The buyTokens function allows users to purchase tokens by sending ETH to the contract. This function calculates the number of tokens to be allocated based on the current ETH price and a predefined rate structure, then transfers the tokens to the buyer. The calcTokensQty function performs the calculation of tokens to be received based on the amount of ETH sent and the current token price in USD.

The getRate function determines the token price based on the number of tokens sold, implementing a tiered pricing model where the price increases as more tokens are sold. The getLatestPrice function fetches the current ETH/USD price from Chainlink's price feed, ensuring that token pricing is up-to-date with market conditions.

Administrative functions include changeActiveStatus, which allows the multisig wallet to start or stop the token sale, and withdraw, which enables the withdrawal of collected ETH to the multisig wallet. Additionally, the recoverTokens function allows the multisig wallet to recover any ERC20 tokens sent to the contract by mistake.

The contract also includes a modifier onlyMultisig to restrict access to certain functions, ensuring that only the multisig wallet can perform critical administrative actions. An event SoldTokens is emitted whenever tokens are sold, providing a log of the transaction on the blockchain.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	AIS	Ascending If Statements	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	ODM	Oracle Decimal Mismatch	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

AIS - Ascending If Statements

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L90
Status	Unresolved

Description

The `getRate` function contains a series of ascending if statements to determine the rate based on the number of tokens sold. Each else if statement includes a check that is already implicitly covered by the preceding conditions. For instance, after confirming that tokensSold is greater than 5,400,000 ether in the first if statement, it is redundant to check this condition again in the subsequent else if statements. This redundancy not only makes the code less readable but also introduces unnecessary checks that can be streamlined.

```
function getRate() public view returns (uint256) {
    if (tokensSold <= 5400000 ether) {
        return 30;
    } else if (
        tokensSold > 5400000 ether && tokensSold <= 10800000
ether
    ) {
        return 31;
    } else if (
        tokensSold > 10800000 ether && tokensSold <= 16200000
ether
    ) {
    ...
}
```

Recommendation

To improve code readability and efficiency, it is recommended to remove the redundant checks within the else if statements. By leveraging the fact that the preceding conditions have already been evaluated, each subsequent else if statement should only verify the upper bound of the range. This will simplify the conditional logic and make the code cleaner and easier to maintain. Ensuring that the conditions are streamlined will enhance the overall efficiency and clarity of the contract.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L147
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion. Specifically the contract owner has the authority to change the state status.

```
function changeActiveStatus(bool isActive) external
onlyMultisig {
    isSaleActive = isActive;
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L156
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. As a result, the users will not be able to find the root cause of the error.

```
require(payable(msg.sender).send(address(this).balance))
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L147
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 changeActiveStatus(bool isActive) external
onlyMultisig {
     isSaleActive = isActive;
}
```

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.

ODM - Oracle Decimal Mismatch

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L78
Status	Unresolved

Description

The contract relies on data retrieved from an external Oracle to make critical calculations. However, the contract does not include a verification step to align the decimal precision of the retrieved data with the precision expected by the contract's internal calculations. This mismatch in decimal precision can introduce substantial errors in calculations involving decimal values.

```
function calcTokensQty(uint256 amountETH) public view returns
(uint256) {
    uint256 ethUsd = getLatestPrice();
    uint256 amountUSD = (amountETH * ethUsd) /
1000000000000000000;
    uint256 tokenPrice = getRate();
    uint256 amountToken = amountUSD / tokenPrice;
    return amountToken * 10 ** 13;
}
```

Recommendation

The team is advised to retrieve the decimals precision from the Oracle API in order to proceed with the appropriate adjustments to the internal decimals representation.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L147
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function changeActiveStatus(bool isActive) external
onlyMultisig {
    isSaleActive = isActive;
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L72
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
function buyTokens() public payable {
    require(isSaleActive, "token sale is not active!");

    uint256 tokens = calcTokensQty(msg.value);
    require(
        token.balanceOf(address(this)) >= tokens,
        "not enough tokens to sell!"
    );

    token.safeTransfer(msg.sender, tokens);
    tokensSold += tokens;

    emit SoldTokens(msg.sender, tokens);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to ensure that the presale tokens are allocated to and held within the contract itself. To determine the appropriate amount of tokens to allocate, the team should calculate the total number of tokens expected to be sold during the entire sale period. Additionally, the total number of sale tokens should be clearly documented and communicated to users. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/TokenSale.sol#L82,83
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 amountToken = amountUSD / tokenPrice
return amountToken * 10 ** 13
```

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.

Functions Analysis

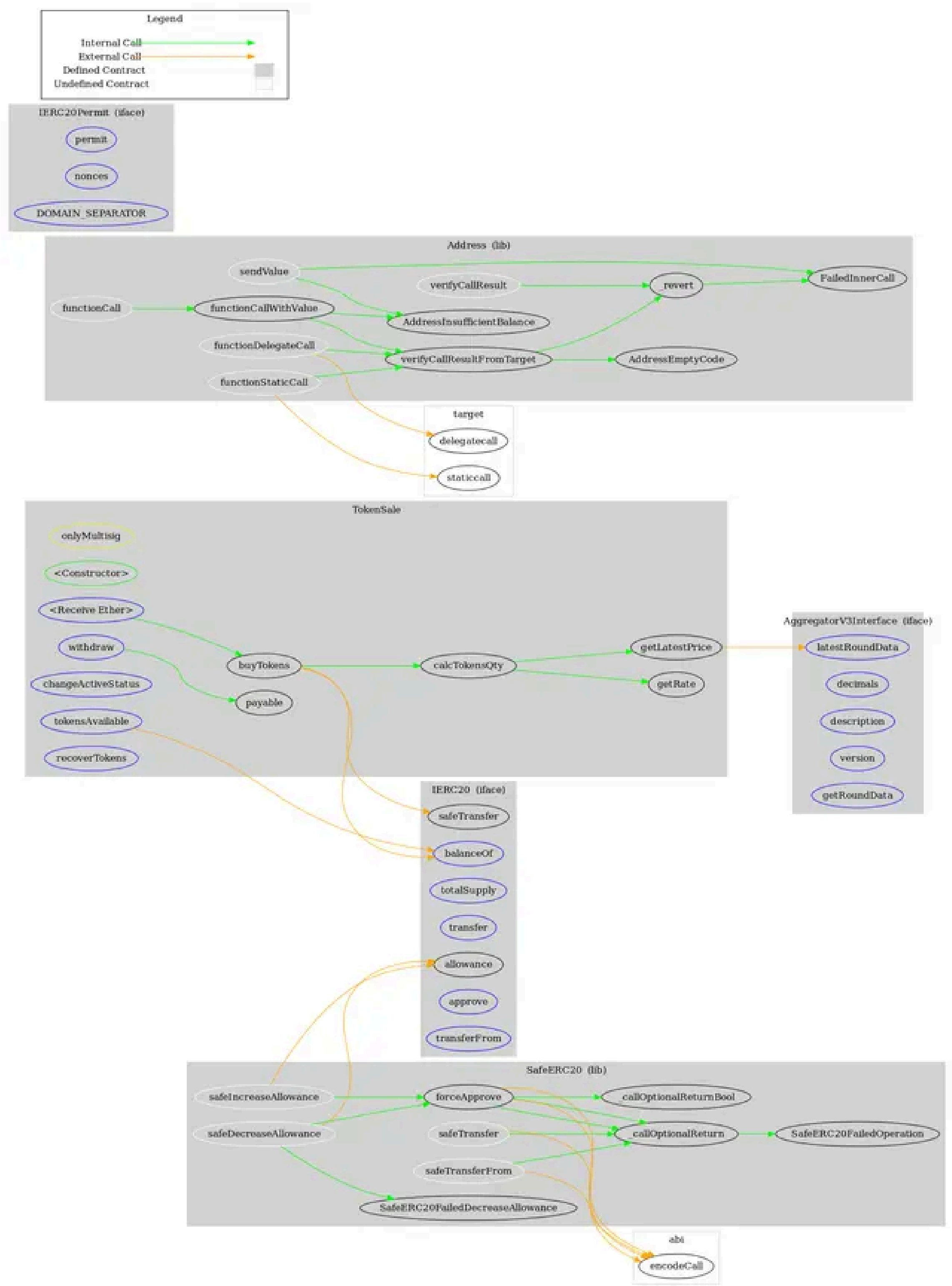
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
TokenSale	Implementation			
		Public	✓	-
		External	Payable	-
	buyTokens	Public	Payable	-
	calcTokensQty	Public		-
	getRate	Public		-
	getLatestPrice	Public		-
	tokensAvailable	External		-
	changeActiveStatus	External	✓	onlyMultisig
	withdraw	External	✓	onlyMultisig
	recoverTokens	External	✓	onlyMultisig
Address	Library			
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		

	verifyCallResult	Internal		
	_revert	Private		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	forceApprove	Internal	✓	
	_callOptionalReturn	Private	✓	
	_callOptionalReturnBool	Private	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-

	DOMAIN_SEPARATOR	External		-
AggregatorV3Interface	Interface			
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
Address (lib)
sendValue
verifyCallResult
revert
FailedInnerCall
functionCall
functionCallWithValue
AddressInsufficientBalance
functionDelegateCall
verifyCallResultFromTarget
AddressEmptyCode
functionStaticCall
target
delegatecall
staticcall
TokenSale
onlyMultisig
<Constructor>
<Receive Ether>
withdraw
buyTokens
calcTokensQty
getLatestPrice
getRate
changeActiveStatus
payable
tokensAvailable
recoverTokens
AggregatorV3Interface (iface)
latestRoundData
decimals
description
version
getRoundData
IERC20 (iface)
safeTransfer
balanceOf
totalSupply
transfer
allowance
approve
transferFrom
SafeERC20 (lib)
safeIncreaseAllowance
forceApprove
callOptionalReturnBool
safeDecreaseAllowance
safeTransfer
callOptionalReturn
SafeERC20FailedOperation
safeTransferFrom
SafeERC20FailedDecreaseAllowance
abi
encodeCall

Summary

Paragon sale contract implements a sales 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