Cyberscope

Audit Report

Paragon Token

June 2024

SHA256 68074aed2f3ab6908ff2ac672d2052a38c7359cf94557bdc07b4baf267d4c923

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	DAE	Duplicate Address Exclusion	Unresolved
	MEE	Missing Events Emission	Unresolved
	MU	Modifiers Usage	Unresolved
	RFC	Redundant Fee Condition	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	OCTD	Transfers Contract's Tokens	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 6
ST - Stops Transactions 7
Description 7
Recommendation 8
ELFM - Exceeds Fees Limit 9
Description 9
Recommendation 10
DAE - Duplicate Address Exclusion 11
Description 11
Recommendation 11
MEE - Missing Events Emission 12
Description 12
Recommendation 12
MU - Modifiers Usage 13
Description 13
Recommendation 13
RFC - Redundant Fee Condition 14
Description 14
Recommendation 15
RSW - Redundant Storage Writes 16
Description 16
Recommendation 16
OCTD - Transfers Contract's Tokens 17
Description 17
Recommendation 18
L19 - Stable Compiler Version 19
Description 19
Recommendation 19
Functions Analysis 20
Inheritance Graph 24
Flow Graph 25
Summary 26
Disclaimer 27

About Cyberscope **28**

Review

Contract Name	ParagonToken
Testing Deploy	https://testnet.bscscan.com/address/0x390a8cc787952c32b3ecd783ecdfbe39576b9a73
Symbol	PRG
Decimals	18
Total Supply	967,750,000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	07 Jun 2024

Source Files

Filename	SHA256
contracts/ParagonToken.sol	68074aed2f3ab6908ff2ac672d2052a38c7359cf94557bdc07b4baf267d4c923
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/ERC20.sol	ddff96777a834b51a08fec26c69bb6ca2d01d150a3142b3fdd8942e07921636a
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939

@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	2e6108a11184dd0caab3f3ef31bd15fed1bc7e4c781a55bc867ccedd8474565c
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81

Findings Breakdown

Critical	2
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/ParagonToken.sol#L83,120,141
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users. The owner may take advantage of it by setting the `maxTxAmount` to zero. As a result, the contract may operate as a honeypot.

```
require(
    value <= maxTxAmount,
    "Transfer amount exceeds the maxTxAmount."
);

function setMaxTxAmount(uint256 amount) external onlyOwner {
    maxTxAmount = amount;
}
```

Recommendation

The contract could embody a check for not allowing setting the `maxTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action. Although the contract suggests that initial owner is a multi-signature wallet, the contract is not yet deployed, so the presence of the multi-signature wallet cannot be verified.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/ParagonToken.sol#L65,74
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setTransferFeePercent` and `setHoldersFeePercent` functions with a high percentage value.

```
function setTransferFeePercent(uint256 value) external
onlyOwner {
    transferFee = value;
}

function setHoldersFeePercent(uint256 value) external onlyOwner
{
    holdersFee = value;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action. Although the contract suggests that initial owner is a multi-signature wallet, the contract is not yet deployed, so the presence of the multi-signature wallet cannot be verified.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

DAE - Duplicate Address Exclusion

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L230,231
Status	Unresolved

Description

The `initWallets` function sets the same address (0x297A7EAafe8F530BC611959992ae2dd29bBd8faA) to be excluded from fees twice, with comments indicating it represents both the Public Sale Wallet and the Community Incentives Wallet. The duplication might be intentional but is undocumented, or it might be a mistake where different addresses were intended. This duplication can cause confusion about the contract's intended behavior and the distribution of tokens.

```
_isExcludedFromFee[0x297A7EAafe8F530BC611959992ae2dd29bBd8faA]
= true; // Public Sale Wallet
_isExcludedFromFee[0x297A7EAafe8F530BC611959992ae2dd29bBd8faA]
= true; // Community Incentives Wallet
```

Recommendation

It is recommended to review and clarify the intended addresses for the Public Sale Wallet and the Community Incentives Wallet. If the same address is intentionally used for both purposes, explicitly document this decision in the contract comments to avoid confusion. If the duplication is not intentional, correct the addresses to ensure they are distinct. Additionally, if the same address is intended for both wallets, removing the redundant assignment to `_isExcludedFromFee` will maintain cleaner and more readable code. Addressing this issue will ensure clearer contract documentation and avoid potential misunderstandings about the token distribution logic.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L38,47,65,74,83,92
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 setTransferFeePercent(uint256 value) external
onlyOwner {
    transferFee = value;
}

function setHoldersFeePercent(uint256 value) external onlyOwner
{
    holdersFee = value;
}

...
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L120,141
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(
        value <= maxTxAmount,
        "Transfer amount exceeds the maxTxAmount."
    );
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

RFC - Redundant Fee Condition

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L161
Status	Unresolved

Description

The `calculateFee` function contains a condition that checks if the `transferFee` is greater than zero before calculating and applying the transfer fee. This condition is redundant and can potentially cause issues. Specifically, if the `transferFee` is set to zero, the function will not proceed to check and apply the `holdersFee`, even if the `holdersFee` is greater than zero. As a result, eligible transactions involving holders might not be charged the intended holders fee, leading to discrepancies in the fee application logic and potentially impacting the tokenomics of the contract.

```
function calculateFee(
        address from,
        address to,
        uint256 value
) internal view returns (uint256) {
    uint256 fee = 0;
    if (
        !_isExcludedFromFee[from] &&
        !_isExcludedFromFee[to] &&
        transferFee > 0
    ) {
        if (checkIsHolder(to)) {
            fee = (value * holdersFee) / 10 ** 4;
        } else {
            fee = (value * transferFee) / 10 ** 4;
        }
    }
    return fee;
}
```

Recommendation

It is recommended to remove the condition that checks if the `transferFee` is greater than zero within the `calculateFee` function. This will ensure that the appropriate fee, either the holdersFee or the `transferFee`, is applied based on the current state and conditions of the transaction. By eliminating this redundant check, the function will correctly handle cases where the `transferFee` is zero but the holdersFee is applicable, thereby maintaining the intended fee structure and logic of the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L38,47,65,74,83,92
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 setTransferFeePercent(uint256 value) external
onlyOwner {
    transferFee = value;
}

function setHoldersFeePercent(uint256 value) external onlyOwner
{
    holdersFee = value;
}

...
```

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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L102
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `getTokens` function.

```
function getTokens(address to, address tokenAddress) external
onlyOwner {
    require(
        tokenAddress != address(0),
        "tokenAddress can not be zero address!"
    );
    uint256 balance =
IERC20(tokenAddress).balanceOf(address(this));
    if (balance > 0) {
        require(
            IERC20(tokenAddress).transfer(to, balance),
            "Token transfer failed"
        );
    }
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action. Although the contract suggests that initial owner is a multi-signature wallet, the contract is not yet deployed, so the presence of the multi-signature wallet cannot be verified.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/ParagonToken.sol#L3
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 ^0.8.20;
```

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

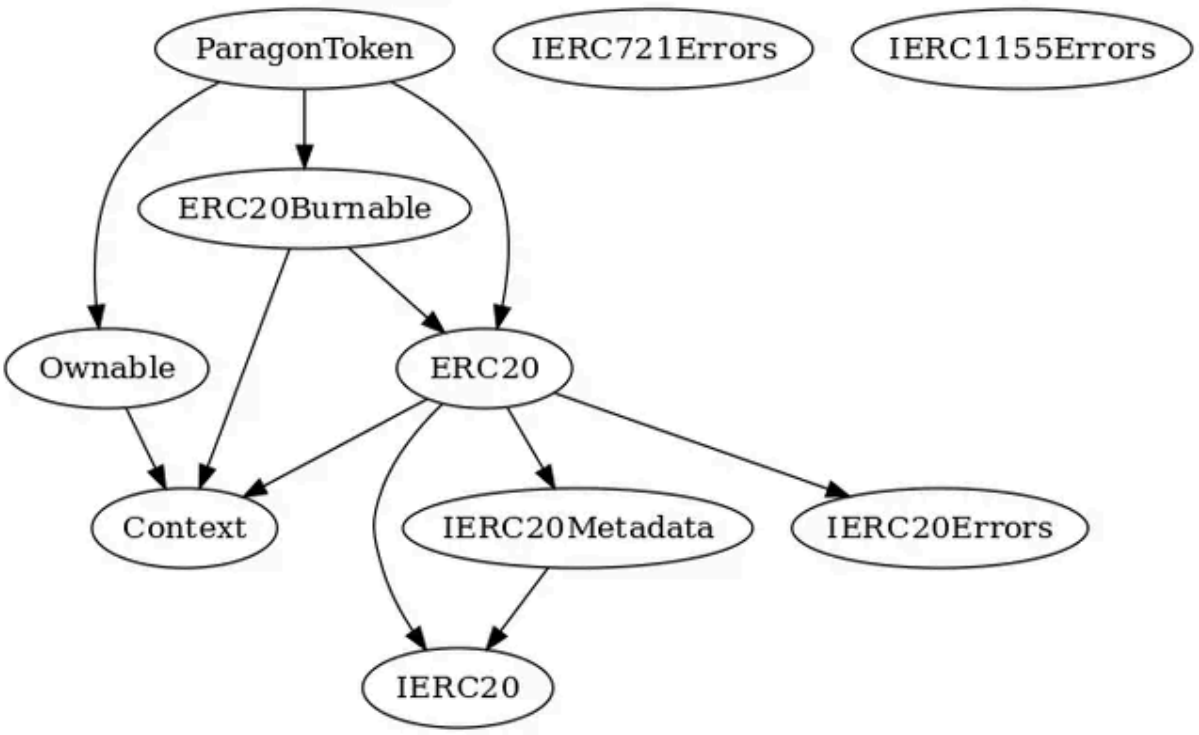
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ParagonToken	Implementation	ERC20, ERC20Burnable, Ownable		
		Public	✓	ERC20 Ownable
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	isExcludedFromFee	Public		-
	setTransferFeePercent	External	✓	onlyOwner
	setHoldersFeePercent	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	setMinHolding	External	✓	onlyOwner
	getTokens	External	✓	onlyOwner
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	calculateFee	Internal		
	checkIsHolder	Public		-
	updateHolders	Internal	✓	
	initWallets	Private	✓	
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ERC20	Implementation	Context, IERC20, IERC20Metadata, IERC20Errors		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-

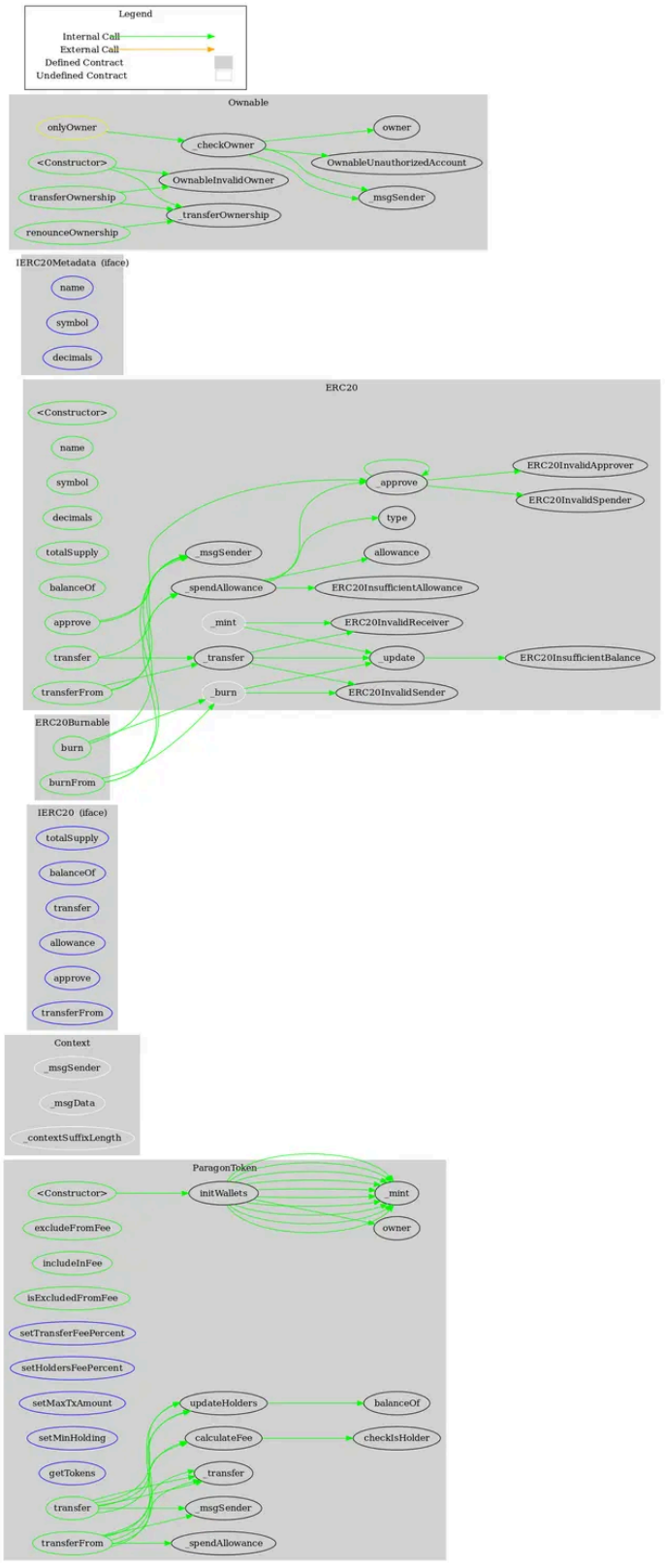
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			

Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
checkOwner
owner
<Constructor>
OwnableInvalidOwner
OwnableUnauthorizedAccount
transferOwnership
msgSender
transferOwnership
renounceOwnership
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transfer
transferFrom
msgSender
spendAllowance
_mint
_transfer
_burn
approve
type
allowance
ERC20InsufficientAllowance
ERC20InvalidReceiver
_update
ERC20InsufficientBalance
ERC20InvalidSender
ERC20InvalidApprover
ERC20InvalidSpender
ERC20Burnable
burn
burnFrom
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
_contextSuffixLength
ParagonToken
<Constructor>
initWallets
_mint
owner
excludeFromFee
includeInFee
isExcludedFromFee
setTransferFeePercent
setHoldersFeePercent
setMaxTxAmount
setMinHolding
getTokens
transfer
transferFrom
updateHolders
calculateFee
transfer
msgSender
spendAllowance
balanceOf
checkIsHolder

Summary

Paragon token 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 manipulate the fees. 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