Cyberscope

Audit Report

PYRAMID

February 2024

Network BSC

Address 0x4F5418446CF1FA75A494c3f0d78eFF4785cE6fB2

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	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IDI	Immutable Declaration Improvement	Unresolved
●	MFN	Misleading Fees Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RES	Redundant Event Statement	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Pyramid
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x4f5418446cf1fa75a494c3f0d78eff4785ce6fb2
Address	0x4f5418446cf1fa75a494c3f0d78eff4785ce6fb2
Network	BSC
Symbol	Pyramid
Decimals	9
Total Supply	100,000,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	25 Feb 2024

Source Files

Filename	SHA256
Pyramid.sol	a90dcd37761ac40d648e0f099e6a0d43e2572d20400eaa232b65cadcc2c57198

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Pyramid.sol#L211,263
Status	Unresolved

Description

The transactions are initially disabled for all users, excluding the authorized addresses. The owner can enable the transactions for all users, by calling the `enableTrading` function. Once the transactions are enabled the owner will not be able to disable them again.

```
function _transfer(address from, address to, uint256 amount)
private {
    require(from != address(0), "ERC20: transfer from the zero
address");
    require(to != address(0), "ERC20: transfer to the zero
address");
    require(amount > 0, "Transfer amount must be greater than
zero");
    uint256 feesum = 0;

    if (!_isExcludedFromFee[from] && !_isExcludedFromFee[to]) {
        require(tradeEnable, "Trading not enabled");
        feesum = amount * buyTaxes / 100;
    }

    if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
        feesum = 0;
    }

    ...
```

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.
- 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Pyramid.sol#L141,142
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
uniswapV2Pair
MarketingWallet
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MFN - Misleading Fees Naming

Criticality	Minor / Informative
Location	Pyramid.sol#L204
Status	Unresolved

Description

The contract employs a fees naming convention `buyTaxes` and `sellTaxes` and associated logic that does not precisely reflect the transaction contexts, leading to potential confusion. First, the application of `buyTaxes` extends beyond traditional buy transactions, affecting any transfer where neither party is excluded from fees, thus misrepresenting the fee's intended context. Second, the logic fails to distinguish between sell transactions initiated by the contract, particularly when the contract is not whitelisted. This oversight results in `buyTaxes` being applied in scenarios where `sellTaxes` might be expected.

```
function _transfer(address from, address to, uint256 amount) private {
    require(from != address(0), "ERC20: transfer from the zero
address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");
    uint256 feesum = 0;

    if (!_isExcludedFromFee[from] && !_isExcludedFromFee[to]) {
        require(tradeEnable, "Trading not enabled");
        feesum = amount * buyTaxes / 100;
    }

    if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
        feesum = 0;
    }

    if (to == uniswapV2Pair && from != address(this) &&
!_isExcludedFromFee[from] && !_isExcludedFromFee[to]) {
        feesum = amount * sellTaxes / 100;

    }

    ...
```

Recommendation

It is recommended to revise the naming conventions of buyTaxes and sellTaxes to more accurately reflect their usage across all transaction types. Additionally, the contract should implement a more refined logic to correctly apply fees based on the transaction's nature and the involved parties.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Pyramid.sol#L282
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 setTaxes(uint256 newBuyFee, uint256 newSellFee)
external onlyOwner {
    require(newBuyFee <= 10 && newSellFee <= 10, "ERC20: wrong
tax value!");
    buyTaxes = newBuyFee;
    sellTaxes = newSellFee;
}
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Pyramid.sol#L243
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tokenAmount) private
lockTheSwap {
    require(tokenAmount > 0, "amount must be greeter than 0");
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router),
tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RES - Redundant Event Statement

Criticality	Minor / Informative
Location	Pyramid.sol#L112
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `FeesUpdated` event statement is not used in the contract's implementation.

```
event FeesUpdated(uint256 indexed _feeAmount);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. It is recommended to remove the unused event statement from the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Pyramid.sol#L282,288
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 setTaxes(uint256 newBuyFee, uint256 newSellFee)
external onlyOwner {
    require(newBuyFee <= 10 && newSellFee <= 10, "ERC20: wrong
tax value!");
    buyTaxes = newBuyFee;
    sellTaxes = newSellFee;
}

function setSwapBackSetting(bool state) external onlyOwner {
    _SwapBackEnable = state;
    emit SwapBackSettingUpdated(state);
}
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Pyramid.sol#L103,102
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private _minSwapTokens = 300000000 * 10**_decimals
uint256 private _maxSwapTokens = 2000000000 * 10**_decimals
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Pyramid.sol#L80,95,97,98,99,100,108,114,295
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.

```
function WETH() external pure returns (address);
address payable private MarketingWallet
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 100000000000 * 10**_decimals
string private constant _name = "Pyramid"
string private constant _symbol = "Pyramid"
bool private _SwapBackEnable = false
event includeFromFeeUpdated(address indexed account);
uint256 _amount
address _tokenAddy
```

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	Pyramid.sol#L284
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.

```
buyTaxes = newBuyFee
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Pyramid.sol#L271,277
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(_isExcludedFromFee[account] != true,"Account is already
excluded")
require(_isExcludedFromFee[account] != false, "Account is
already included")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Pyramid.sol#L299
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.

```
IERC20(_tokenAddy).transfer(MarketingWallet, _amount)
```

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.

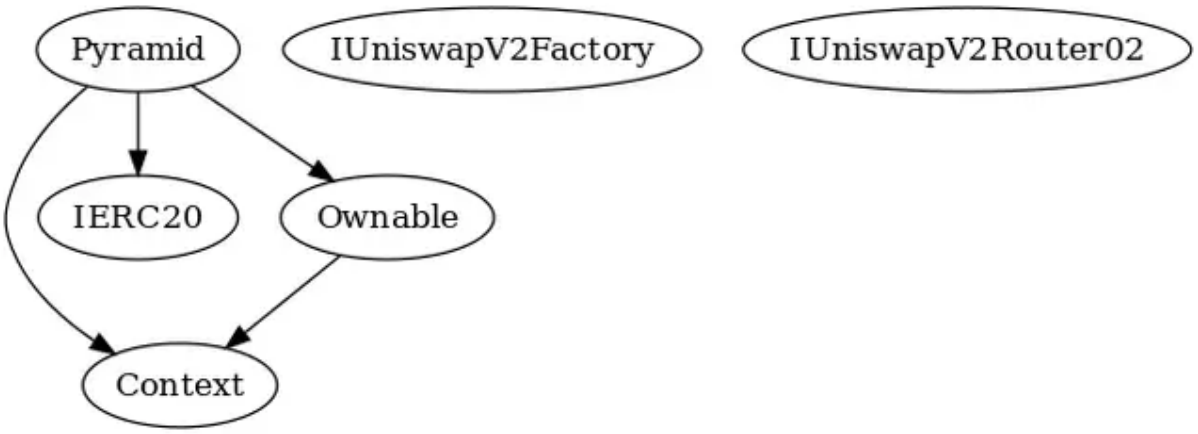
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
	renounceOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			

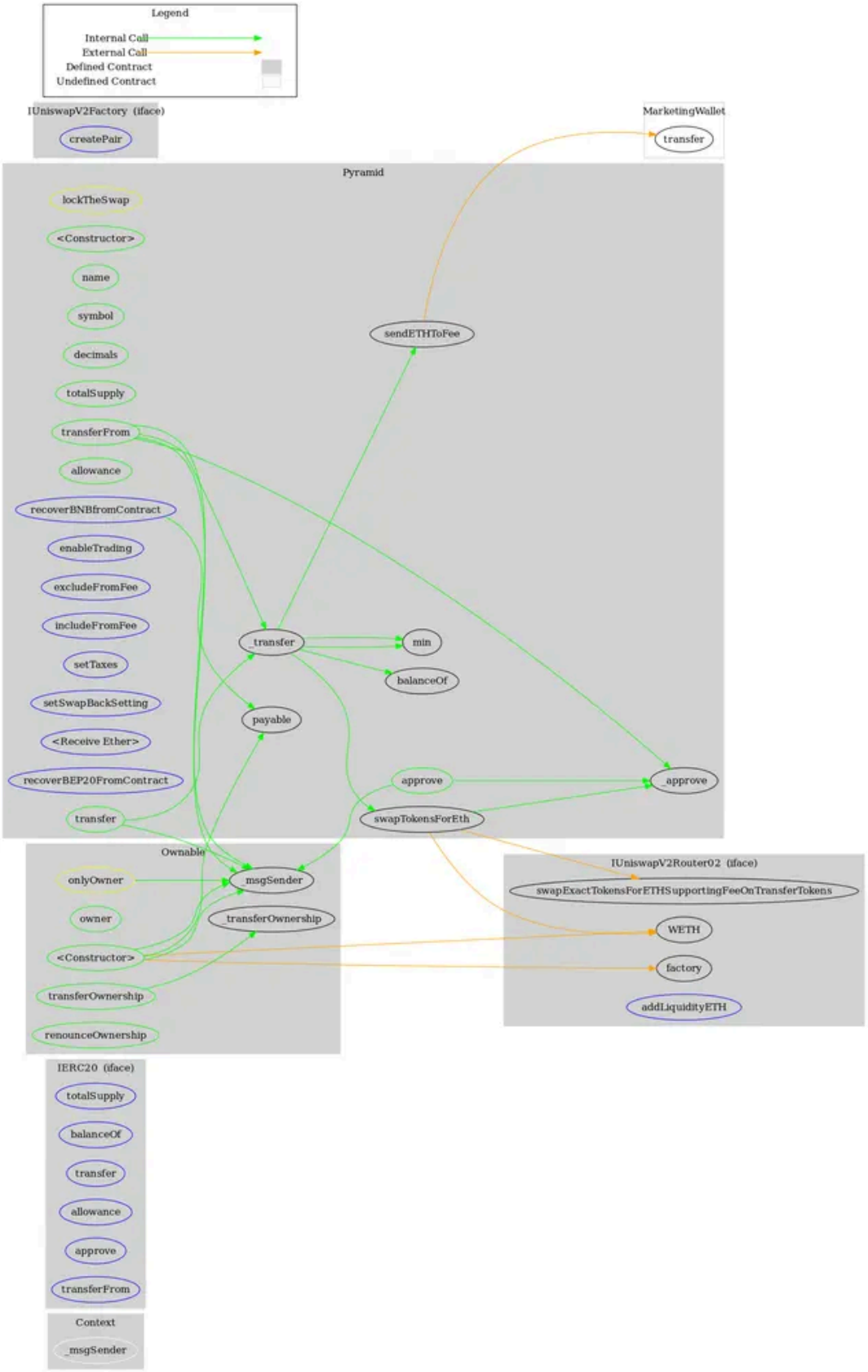
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
Pyramid	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	min	Private		
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	

	swapTokensForEth	Private	✓	lockTheSwap
	sendETHToFee	Private	✓	
	enableTrading	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	includeFromFee	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setSwapBackSetting	External	✓	onlyOwner
		External	Payable	-
	recoverBEP20FromContract	External	✓	onlyOwner
	recoverBNBfromContract	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
MarketingWallet
transfer
Pyramid
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
transferFrom
allowance
recoverBNBfromContract
enableTrading
excludeFromFee
includeFromFee
setTaxes
setSwapBackSetting
<Receive Ether>
recoverBEP20FromContract
transfer
sendETHToFee
_transfer
min
balanceOf
payable
approve
approve
swapTokensForEth
Ownable
onlyOwner
owner
<Constructor>
transferOwnership
renounceOwnership
_msgSender
transferOwnership
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

PYRAMID contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The transactions are initially disabled and they have to be enabled by the contract owner.

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