Cyberscope

Audit Report

Thoon

February 2024

Network	BSC
Address	0xc460457cb915c6ed96bf297db603c0b6d838fea7
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
●	CR	Code Repetition	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MVN	Misleading Variable Naming	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RC	Redundant Calculation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	UOD	Unnecessary Override Declaration	Unresolved
●	UZA	Unnecessary Zero Addition	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

Table of Contents

Review

Contract Name	Thoon
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xc460457cb915c6ed96bf297db603c0b6d838fea7
Address	0xc460457cb915c6ed96bf297db603c0b6d838fea7
Network	BSC
Symbol	$thoon
Decimals	18
Total Supply	21,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	26 Feb 2024

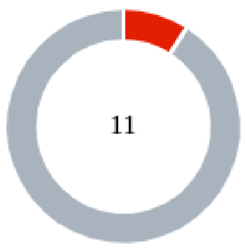
Source Files

Filename	SHA256
Token.sol	6e23fbb16bcf86b997cfd1adf9c6d3ed9913bfa4252fc7b5a3a0468a2bf1fdde
Strings.sol	d81a4a55da7145b070af1100e65d0437bee08606024266d5c39f116c1fd6ca47

StorageSlot.sol	51c993d87ee6096af06f8701d8497272751c8cc9c9c2b31e67a3087d5f2302df
SignedMath.sol	420a5a5d8d94611a04b39d6cf5f02492552ed4257ea82aba3c765b1ad52f77f6
ShortStrings.sol	0f1abe1770f7e80f0b9221aac780424a42e5c64b9814aa735d28115a0f549915
Ownable2Step.sol	3e3bdb084bc14ade54e8259e710287956a7dbf2b2b4ad1e4cd8899d2293c7241
Ownable.sol	33422e7771fefe5fbfe8934837515097119d82a50eda0e49b38e4d6a64a1c25d
Math.sol	85a2caf3bd06579fb55236398c1321e15fd524a8fe140dff748c0f73d7a52345
Initializable.sol	b05c26d897c4178cbdb35ad113527e463e1bdeae5764869318a54f93c8b98a94
IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
IUniswapV2Pair.sol	29c75e69ce173ff8b498584700fef76bc81498c1d98120e2877a1439f0c31b5a
IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
IERC5267.sol	72424d3a7a9b7d1c3584d4ec83bf116f30dfaf7180298a32669eefacb55ccb92
IERC20Permit.sol	b7383c48331f3cc9901fc05e5d5830fcd533699a77f3ee1e756a98681bfbb2ee
IERC20Metadata.sol	b10e2f8bcc3ed53a5d9a82a29b1ad3209225331bb4de4a0459862a762cf83a1a

IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
ERC20Permit.sol	0480d1a7597feaaaff415319b409cabf03f8eda3f10e6c2db79122343a1aef36
ERC20Burnable.sol	480b22ce348050fdb85a693e38ed6b4767a94e4776fc6806d6808a0ec171177e
ERC20.sol	f70c6ae5f2dda91a37e17cfcbec390cc59515ed0d34e316f036f5431b5c0a3f2
EIP712.sol	49e55a46cc36d9c820b7081ac3739f2fe177b16ab18277561688b75ad56b42f2
ECDSA.sol	9de720b838a1cc4ef28dfb89287e7422238d7aace4213ce6aea699392b6879e7
Counters.sol	2fdcb1343e5621385b62e57b5c7775607c272122b6f2dc77da8f84828aa40cd0
Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	10

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Token.sol#L346,367
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again. Additionally, the contract incorporates an anti-bot mechanism that restricts users not excluded from limits from transacting more frequently than a specified cooldown period, up to 12 hours. While this feature aims to mitigate abusive trading practices, the lack of flexibility in re-disabling transactions or adjusting the anti-bot mechanism parameters post-enablement could limit the contract's adaptability to evolving security threats or operational requirements.

```
  function _beforeTokenTransfer(address from, address to, uint256
amount)
        internal
        override
    {
        if(!isExcludedFromLimits[from])
            require(lastTrade[from] + tradeCooldownTime <=
block.timestamp, "Antibot: Transaction sender is in anti-bot
cooldown");
        if(!isExcludedFromLimits[to])
            require(lastTrade[to] + tradeCooldownTime <=
block.timestamp, "Antibot: Transaction recipient is in anti-bot
cooldown");

        // Interactions with DEX are disallowed prior to enabling
trading by owner
        if ((AMMPairs[from] && !isExcludedFromTradingRestriction[to])
|| (AMMPairs[to] && !isExcludedFromTradingRestriction[from])) {
            require(tradingEnabled, "EnableTrading: Trading was not
enabled yet");
        }

        super._beforeTokenTransfer(from, to, amount);
    }
```

```
   function updateTradeCooldownTime(uint256 _tradeCooldownTime)
public onlyOwner {
        require(_tradeCooldownTime <= 12 hours, "Antibot: Trade
cooldown too long");

        tradeCooldownTime = _tradeCooldownTime;

        emit TradeCooldownTimeUpdated(_tradeCooldownTime);
    }
        function enableTrading() external onlyOwner {
        require(!tradingEnabled, "EnableTrading: Trading was enabled
already");
        tradingEnabled = true;

        emit TradingEnabled();
    }
```

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 multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

CR - Code Repetition

Criticality	Minor / Informative
Location	Token.sol#L158,169,214
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.

```
    function marketingFeesSetup(uint16 _buyFee, uint16 _sellFee, uint16 
_transferFee) public onlyOwner {
        totalFees[0] = totalFees[0] - marketingFees[0] + _buyFee;
        totalFees[1] = totalFees[1] - marketingFees[1] + _sellFee;
        totalFees[2] = totalFees[2] - marketingFees[2] + _transferFee;
        require(totalFees[0] <= 2500 && totalFees[1] <= 2500 && 
totalFees[2] <= 2500, "TaxesDefaultRouter: Cannot exceed max total fee 
of 25%");

        marketingFees = [_buyFee, _sellFee, _transferFee];

        emit marketingFeesUpdated(_buyFee, _sellFee, _transferFee);
    }

    function autoBurnFeesSetup(uint16 _buyFee, uint16 _sellFee, uint16 
_transferFee) public onlyOwner {
        totalFees[0] = totalFees[0] - autoBurnFees[0] + _buyFee;
        totalFees[1] = totalFees[1] - autoBurnFees[1] + _sellFee;
        totalFees[2] = totalFees[2] - autoBurnFees[2] + _transferFee;
        require(totalFees[0] <= 2500 && totalFees[1] <= 2500 && 
totalFees[2] <= 2500, "TaxesDefaultRouter: Cannot exceed max total fee 
of 25%");

        autoBurnFees = [_buyFee, _sellFee, _transferFee];

        emit autoBurnFeesUpdated(_buyFee, _sellFee, _transferFee);
    }

    function liquidityFeesSetup(uint16 _buyFee, uint16 _sellFee, uint16 
_transferFee) public onlyOwner {
        ...
    }
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Token.sol#L255
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
_marketingPending += fees * marketingFees[txType] / totalFees[txType];

if (autoBurnFees[txType] > 0) {
    autoBurnPortion = fees * autoBurnFees[txType] / totalFees[txType];
    _burn(from, autoBurnPortion);
    emit autoBurned(autoBurnPortion);
}

    _liquidityPending += fees * liquidityFees[txType] /
totalFees[txType];
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

MVN - Misleading Variable Naming

Criticality	Minor / Informative
Location	Token.sol#L282
Status	Unresolved

Description

The contract is implementing a functionality intended to swap tokens for a certain amount of Ether (ETH) and subsequently update a variable to reflect the amount of ETH received as a result of this swap. The `coinsReceived` variable, is intended to store the amount of ETH that the contract has accumulated. However, the way it is currently implemented, `coinsReceived = address(this).balance;` , inaccurately suggests that it solely represents the ETH received from the most recent swap. This is misleading because the assignment is made post-swap without accounting for the contract's balance prior to the swap, thereby causing `coinsReceived` to represent the total balance of the contract, which includes both the initial balance before the swap and the additional ETH received from the swap.

```
_swapTokensForCoin(token2Swap);
uint256 coinsReceived = address(this).balance;
```

Recommendation

It is recommended to rename the variable to more accurately reflect its actual functionality. Instead of using `coinsReceived = address(this).balance;` , the implementation should calculate the balance before the swap and then add the newly received amount to it. A more appropriate approach would involve initializing a variable to store the contract's balance before the swap and then calculating the difference after the swap to obtain the actual amount of coins received. This process will ensure that the variable name and its value accurately represent the additional amount of ETH received from the swap, thereby improving code readability and maintainability.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Token.sol#L123
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 _swapTokensForCoin(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = routerV2.WETH();

        _approve(address(this), address(routerV2), tokenAmount);


routerV2.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAm
ount, 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.

RC - Redundant Calculation

Criticality	Minor / Informative
Location	Token.sol#L285
Status	Unresolved

Description

The contract is designed to swap a certain amount of tokens, denoted as `_marketingPending`, for Ether (ETH) and subsequently calculate a portion of the received ETH to allocate for marketing purposes. The calculation for the marketing portion is performed using the formula `marketingPortion = coinsReceived * _marketingPending / token2Swap;`. However, the assignment `token2Swap = 0 + _marketingPending;` directly before this calculation implies that `token2Swap` is always equal to `_marketingPending`. Given this equality, the formula simplifies to `marketingPortion = coinsReceived * _marketingPending / _marketingPending;`, effectively rendering the multiplication and division by the same amount (`_marketingPending`) unnecessary. As a result, `marketingPortion` will always equal `coinsReceived`, indicating that the entire amount of coins received from the swap is being allocated for marketing, which makes the calculation redundant.

```
if (false || _marketingPending > 0) {
    uint256 token2Swap = 0 + _marketingPending;
     bool success = false;

    _swapTokensForCoin(token2Swap);
    uint256 coinsReceived = address(this).balance;

    uint256 marketingPortion = coinsReceived * _marketingPending /
token2Swap;
    if (marketingPortion > 0) {
```

Recommendation

It is recommended to eliminate the redundant calculation to enhance code efficiency and clarity. Since `token2Swap` is equal to `_marketingPending`, and the calculation of `marketingPortion` results in it being equivalent to `coinsReceived`, the explicit calculation can be bypassed. Instead, directly assign `coinsReceived` to `marketingPortion` when appropriate, or reconsider the logic if the intention was to allocate only a portion of `coinsReceived` for marketing. This simplification will not only reduce computational overhead but also improve the readability of the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Token.sol#L225
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 excludeFromFees(address account, bool isExcluded)
public onlyOwner {
        isExcludedFromFees[account] = isExcluded;

        emit ExcludeFromFees(account, isExcluded);
    }
```

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.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	Token.sol#L185,282
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
_swapTokensForCoin(halfAmount);
...
_swapTokensForCoin(token2Swap);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

UOD - Unnecessary Override Declaration

Criticality	Minor / Informative
Location	Token.sol#L119
Status	Unresolved

Description

The contract is currently implementing an override of the decimals function, which simply returns the value 18. This override is redundant since the extending token contract already specifies 18 decimals as its standard. In the context of ERC-20 tokens, 18 decimals is a common default, and overriding this function to return the same value adds unnecessary complexity to the contract. This redundancy does not contribute to the functionality of the contract and could potentially lead to confusion about the necessity of this override.

```
    function decimals() public pure override returns (uint8) {
        return 18;
    }
```

Recommendation

It is recommended to remove the `override` keyword from the decimals function, assuming the extending token contract already defines 18 decimals. This action will simplify the contract by eliminating redundant code, thereby enhancing its clarity and efficiency. The removal of this unnecessary override will not impact the contract's functionality but will contribute to a cleaner and more maintainable codebase.

UZA - Unnecessary Zero Addition

Criticality	Minor / Informative
Location	Token.sol#L145
Status	Unresolved

Description

The contract is utilizing the `getAllPending` function designed to calculate the total amount of pending marketing and liquidity fees. Within this function, the total is calculated using the expression `return 0 + _marketingPending + _liquidityPending;` . The addition of zero at the beginning of this expression is redundant and does not affect the outcome of the calculation. In programming, adding zero to a number does not change its value, making this part of the expression superfluous. This unnecessary operation could potentially lead to confusion about the code's intent and unnecessarily clutters the expression, detracting from the overall clarity and simplicity of the contract's code.

```
    function getAllPending() public view returns (uint256) {
        return 0 + _marketingPending + _liquidityPending;
    }
```

Recommendation

It is recommended to remove the redundant zero addition from the expression in the `getAllPending` function to improve code clarity and conciseness. The corrected expression should simply be return `_marketingPending + _liquidityPending;` . This change not only adheres to best coding practices by eliminating unnecessary operations but also enhances the readability of the code, making it easier for other users to understand the contract's functionality. Simplifying expressions where possible is a key aspect of writing clean, efficient, and easily maintainable code in smart contract development.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Token.sol#L44,56,57,58,60,61,63,64,113,133,148,158,169,214,346
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.

```
mapping (address => bool) public AMMPairs;
event autoBurnFeesUpdated(uint16 buyFee, uint16 sellFee, uint16
transferFee);
event autoBurned(uint256 amount);
...
```

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

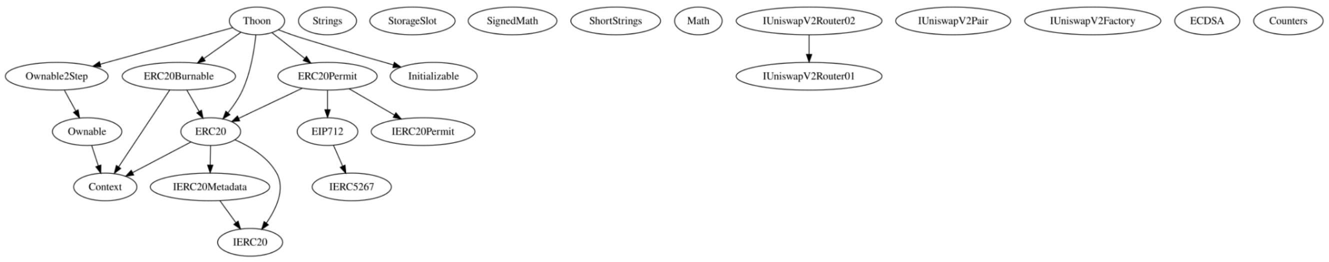
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Thoon	Implementation	ERC20, ERC20Burnable, Ownable2Step, ERC20Permit, Initializable		
		Public	✓	ERC20 ERC20Permit
	initialize	External	✓	initializer
		External	Payable	-
	decimals	Public		-
	_swapTokensForCoin	Private	✓	
	updateSwapThreshold	Public	✓	onlyOwner
	getSwapThresholdAmount	Public		-
	getAllPending	Public		-
	marketingAddressSetup	Public	✓	onlyOwner
	marketingFeesSetup	Public	✓	onlyOwner
	autoBurnFeesSetup	Public	✓	onlyOwner
	_swapAndLiquify	Private	✓	
	_addLiquidity	Private	✓	
	addLiquidityFromLeftoverTokens	External	✓	-
	liquidityFeesSetup	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner

	_transfer	Internal	✓	
	_updateRouterV2	Private	✓	
	setAMMPair	External	✓	onlyOwner
	_setAMMPair	Private	✓	
	excludeFromLimits	External	✓	onlyOwner
	_excludeFromLimits	Internal	✓	
	updateTradeCooldownTime	Public	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	excludeFromTradingRestriction	Public	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	

Inheritance Graph



Flow Graph

Summary

Thoon 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. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% fees.

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