Cyberscope

Audit Report

Grok New Year

December 2023

Network BSC

Address 0x6bdB939815164d9107aEBa34EEe962BA0EFCC158

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
	PVC	Price Volatility Concern	Unresolved
	DDP	Decimal Division Precision	Unresolved
	MAU	Misleading Address Usage	Unresolved
	CR	Code Repetition	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	PAV	Pair Address Validation	Unresolved
	MEE	Missing Events Emission	Unresolved
	MVD	Misleading Variable Definition	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	GrokNY
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	99999 runs
Explorer	https://bscscan.com/address/0x6bdb939815164d9107aeba34eee962ba0efcc158
Address	0x6bdb939815164d9107aeba34eee962ba0efcc158
Network	BSC
Symbol	GrokNY
Decimals	18
Total Supply	10,000,000,000

Audit Updates

Initial Audit	23 Dec 2023

Source Files

Filename	SHA256
contracts/GrokNYDistribution.sol	9dd5491ee2929d87459fbe2f1120ccb3aaccac2bd128ebbb4d9926f8a8c94446
contracts/GrokNY.sol	dc1200a796f275d6934c4868ab60a21a4bfdfa39eee2aa7f72651f65d1bc2486
contracts/interfaces/IGrokNYDistribution.sol	3cef3dd78bad33c8f2c7b13785baa081cdbd844901aae606de1a34a2c407c376

@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
@openzeppelin/contracts/utils/Strings.sol	cb2df477077a5963ab50a52768cb74ec6f32177177a78611ddbbe2c07e2d36de
@openzeppelin/contracts/utils/Context.sol	b2cfee351bcafd0f8f27c72d76c054df9b571b62cfac4781ed12c86354e2a56c
@openzeppelin/contracts/utils/math/SignedMath.sol	420a5a5d8d94611a04b39d6cf5f02492552ed4257ea82aba3c765b1ad52f77f6
@openzeppelin/contracts/utils/math/Math.sol	85a2caf3bd06579fb55236398c1321e15fd524a8fe140dff748c0f73d7a52345
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/access/IAccessControl.sol	d03c1257f2094da6c86efa7aa09c1c07ebd33dd31046480c5097bc2542140e45
@openzeppelin/contracts/access/AccessControl.sol	afd98330d27bddff0db7cb8fcf42bd4766dda5f60b40871a3bec6220f9c9edf7

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/GrokNY.sol#L360
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 enabled the owner will not be able to disable them again.

```
require(tradingEnabled || excludedFromFee[_from] || excludedFromFee[_to],
"ERC20: Trading not yet enabled");
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L272
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `feeLimit` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH. This poses a significant risk, especially if the swap amount exceeds the router's slippage threshold. Such a scenario could result in the failure of the sale transaction.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setLimit(uint256 _feeLimit) external onlyRole(DEFAULT_ADMIN_ROLE) {
    feeLimit = _feeLimit;

    emit FeeLimitUpdated();
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L455,476,484
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.

```
uint256 _liquidityFeeToken1Amount = _calcFee(_token1Balance,
_liquidityFeeHalf * 10000 / _amountToSwap);
uint256 _swapFeeToken1Amount = _calcFee(_token1Balance, swapFeeAmount *
10000 / _amountToSwap);
uint256 _rewardToken1Amount = _calcFee(_token1Balance, _rewardsToSwap *
10000 / _amountToSwap);
```

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.

MAU - Misleading Address Usage

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L402
Status	Unresolved

Description

The contract contains an array of addresses called `burnFeeReceivers` to represent a specific type of address, commonly acknowledged in the blockchain for a particular purpose. However, these wallet addresses within this contract are mutable, meaning they can be altered. As a result, the designated addresses may not consistently serve their conventional purpose, potentially leading to unintended behaviors within the contract's operation. This mutable design diverges from the standard practice of utilizing a fixed, immutable address for such purposes, thereby introducing a layer of complexity and potential risk in the contract's functionality.

```
if (burnFeeReceivers.length > 0) {
    for (uint256 _i = 0; _i < burnFeeReceivers.length; _i++) {
        _transferAmount(_from, burnFeeReceivers[_i], _calcFee(_burnFeeRes,
burnFeeReceiversRate[_i]));
    }
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the clarity and comprehensibility of the code to ensure that it accurately reflects the intended functionality. The designated address, which reflects a specific purpose within the contract, should ideally be immutable to maintain consistency in its functionality and to adhere to common practices, thereby reducing the potential for unexpected behaviors or vulnerabilities within the contract's operation.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L198,281,301,327
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.

```
uint256 _totalRate = 0;
for (uint256 _i = 0; _i < _rewardSwapReceiversRate.length; _i++) {
    _totalRate += _rewardSwapReceiversRate[_i];
}
require(_totalRate == 10000, "rate");
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L178,184,190,351
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.

```
isLpToken[_lpToken] = _lp;
excludedFromFee[_address] = _isExcludedFromFee;
excludedFromSwap[_address] = _isExcludedFromSwap;
enabledSwapForSell = _enabledSwapForSell;
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L174
Status	Unresolved

Description

The `setLpToken` function allows the admin role to set any arbitrary value without validation to the `isLpToken` mapping, which is supposed to hold Uniswap pair addresses. This lack of validation can lead to unintended behavior, including the potential disruption of the contract's intended functionality.

```
function setLpToken(address _lpToken, bool _lp) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_lpToken != address(0), "ERC20: invalid LP address");
    require(_lpToken != pair, "ERC20: exclude default pair");

    isLpToken[_lpToken] = _lp;

    emit LpTokenUpdated(_lpToken, _lp);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L171
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.

```
tradingEnabled = true;
```

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.

MVD - Misleading Variable Definition

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L22
Status	Unresolved

Description

The contract initializes the `owner` variable to the zero address (`address(0)`), which might create a misleading impression that the contract is renounced from its creation. However, the contract utilizes the AccessControl contract from OpenZeppelin, allowing the owner to have administrative rights and perform certain functions. This discrepancy in the initialization of the owner variable could be confusing for users and developers.

```
address public owner = address(0);
```

Recommendation

The team should consider updating the initialization of the owner variable to match the actual owner address or provide a clear comment explaining the use of the AccessControl contract and the role designated as the owner. This will help avoid potential confusion and provide transparency regarding the contract's ownership structure.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L22
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.

```
address public owner = address(0)
```

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	contracts/GrokNYDistribution.sol#L14contracts/GrokNY.sol#L127,131,136,140,145,155,160,174,183,189,195,215,222,236,246,256,272,278,298,324,350
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 _token
address _to
uint256 _amount
address _account
address _recipient
address _owner
address _spender
address _sender
uint256 _addedValue
uint256 _subtractedValue
address _lpToken
bool _lp
address _address
bool _isExcludedFromFee

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/GrokNY.sol#L426,455
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 _liquidityFeeHalf = liquidityFeeAmount / 2
uint256 _liquidityFeeToken1Amount = _calcFee(_token1Balance,
_liquidityFeeHalf * 10000 / _amountToSwap)
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/interfaces/IGrokNYDistribution.sol#L2contracts/GrokNYDistribution.sol#L2contracts/GrokNY.sol#L2
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 ^0.8.2;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/GrokNYDistribution.sol#L15
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(_token).transfer(_to, _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.

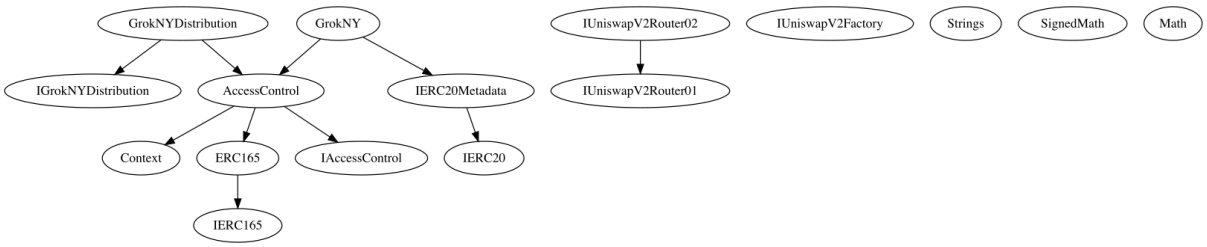
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
GrokNYDistribution	Implementation	IGrokNYDistribution, AccessControl		
		Public	✓	-
	recoverTokensFor	External	✓	onlyRole
GrokNY	Implementation	IERC20Metadata, AccessControl		
		Public	✓	-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	Public		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	enableTrading	External	✓	onlyRole
	setLpToken	External	✓	onlyRole
	setExcludedFromFee	Public	✓	onlyRole

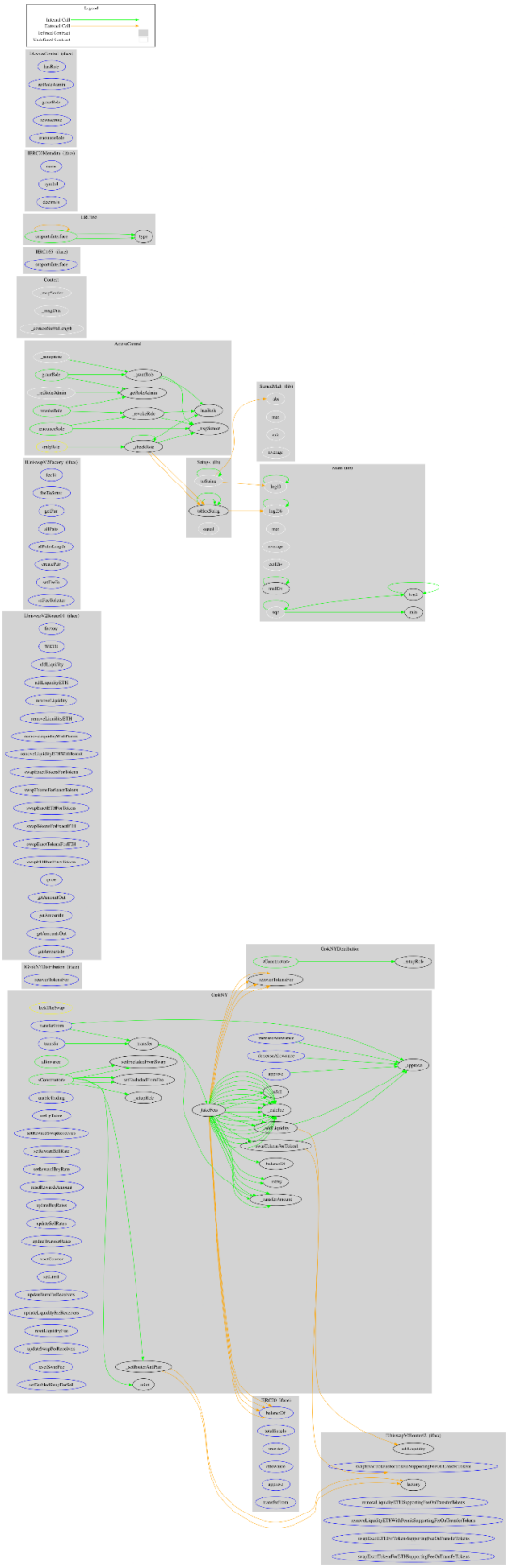
	setExcludedFromSwap	Public	✓	onlyRole
	setRewardSwapReceivers	External	✓	onlyRole
	setRewardSellRate	External	✓	onlyRole
	setRewardBuyRate	External	✓	onlyRole
	resetRewardsAmount	External	✓	onlyRole
	updateBuyRates	External	✓	onlyRole
	updateSellRates	External	✓	onlyRole
	updateTransferRates	External	✓	onlyRole
	resetCounter	External	✓	onlyRole
	setLimit	External	✓	onlyRole
	updateBurnFeeReceivers	External	✓	onlyRole
	updateLiquidityFeeReceivers	External	✓	onlyRole
	resetLiquidityFee	External	✓	onlyRole
	updateSwapFeeReceivers	External	✓	onlyRole
	resetSwapFee	External	✓	onlyRole
	setEnabledSwapForSell	External	✓	onlyRole
	_transfer	Internal	✓	
	_takeFees	Internal	✓	
	_transferAmount	Internal	✓	
	_mint	Internal	✓	
	_approve	Internal	✓	
	_setRouterAndPair	Internal	✓	
	_calcFee	Internal		

	_isSell	Internal		
	_isBuy	Internal		
	_swapTokensForToken1	Internal	✓	lockTheSwap
	_addLiquidity	Internal	✓	lockTheSwap
IGrokNYDistribution	Interface			
	recoverTokensFor	External	✓	-

Inheritance Graph



Flow Graph



Summary

Grok New Year 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 stopping transactions. A multi-wallet signing pattern will provide security against potential hacks. There is also a limit of max 9% 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