Cyberscope

Audit Report

MASTERNODED

August 2023

Network	ETH
Address	0xCa93A5d889e445CECb42E5386f7d516511d2820f
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	RSK	Redundant Storage Keyword	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	MasternodedToken
Compiler Version	v0.8.21+commit.d9974bed
Optimization	200 runs
Explorer	https://etherscan.io/address/0xca93a5d889e445cecb42e5386f7d516511d2820f
Address	0xca93a5d889e445cecb42e5386f7d516511d2820f
Network	ETH
Symbol	NODED
Decimals	18
Total Supply	1,000,000,000,000

Audit Updates

Initial Audit	25 Aug 2023

Source Files

Filename	SHA256
MasternodedToken.sol	aed3fa32d19d8bc347726e365d51671a3362c8e8d8502c549a97d854a326b3be

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	MasternodedToken.sol#L1254,1371,1381,1391,1401,1411,1421,1421,1431,1441,1441,1550,1564,3192,3360,3360
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Counter storage counter
AddressSlot storage r
BooleanSlot storage r
Bytes32Slot storage r
Uint256Slot storage r
StringSlot storage r
string storage store
BytesSlot storage r
bytes storage store
Checkpoint[] storage ckpts
Checkpoint storage result
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MasternodedToken.sol#L1158,2471,3016,3060,3127
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 CLOCK_MODE() external view returns (string memory);

n DOMAIN_SEPARATOR() external view returns (bytes32);
}



...


n DOMAIN_SEPARATOR() external view override returns (bytes32) {
        return _domainSeparatorV4();
    }



...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	MasternodedToken.sol#L38,55,72,106,123,140,157,174,191,208,225,242,259,276,293,310,327,344,361,378,395,412,429,446,480,514,531,548,562,580,598,616,634,652,670,688,706,724,742,760,778,796,814,832,850,868,886,904,922,940,958,976,994,1012,1030,1048,1066,1084,1102,1120,1134,1264,1272,1371,1381,1391,1401,1411,1431,1441,1564,1587,1594,1602,1611,1639,1665,1675,1759,1808,1861,1872,1910,1923,1953,1980,2005,2012,2021,2036,2043,2103,2136,2149,2160,2213,2232,2262,3349,3353
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function toUint248(uint256 value) internal pure returns
(uint248) {
        require(value <= type(uint248).max, "SafeCast: value
doesn't fit in 248 bits");
        return uint248(value);
    }

function toUint240(uint256 value) internal pure returns
(uint240) {
        require(value <= type(uint240).max, "SafeCast: value
doesn't fit in 240 bits");
        return uint240(value);
    }

function toUint232(uint256 value) internal pure returns
(uint232) {
        require(value <= type(uint232).max, "SafeCast: value
doesn't fit in 232 bits");
        return uint232(value);
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MasternodedToken.sol#L1721,1724,1736,1740,1741,1742,1743,1744,1745,1751
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.

```
denominator := div(denominator, twos)
inverse *= 2 - denominator * inverse
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MasternodedToken.sol#L3322
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
oldWeight, uint2
 newWeight) = _wr
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MasternodedToken.sol#L3023
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
memory name) EIP71
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	MasternodedToken.sol#L1373,1383,1393,1403,1413,1423,1433,1443,1517,1682,1986,2111,2217,2247,3361
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            r.slot := slot
        }

assembly {
            r.slot := store.slot
        }

assembly {
            mstore(str, len)
            mstore(add(str, 0x20), sstr)
        }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	MasternodedToken.sol#L8,1146,1165,1225,1236,1282,1314,1454,1578,1624,1966,2053,2272,2416,2479,2506,2587,2617,2984,3081,3373,3412
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
ma solidity ^0.8.0;


solidity ^0.8.0;

...


solidity ^0.8.9;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MasternodedToken.sol#L8,1146,1165,1225,1236,1282,1314,1454,1578,1624,1966,2053,2272,2416,2479,2506,2587,2617,2984,3081,3373,3412
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 ^0.8.0;
pragma solidity ^0.8.8;

ma solidity ^0.8.0;



solidity ^0.8.8;



solidity ^0.8.0;

/**

...
```

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
SafeCast	Library			
	toUint248	Internal		
	toUint240	Internal		
	toUint232	Internal		
	toUint224	Internal		
	toUint216	Internal		
	toUint208	Internal		
	toUint200	Internal		
	toUint192	Internal		
	toUint184	Internal		
	toUint176	Internal		
	toUint168	Internal		
	toUint160	Internal		
	toUint152	Internal		
	toUint144	Internal		
	toUint136	Internal		
	toUint128	Internal		
	toUint120	Internal		

	toUint112	Internal		
	toUint104	Internal		
	toUint96	Internal		
	toUint88	Internal		
	toUint80	Internal		
	toUint72	Internal		
	toUint64	Internal		
	toUint56	Internal		
	toUint48	Internal		
	toUint40	Internal		
	toUint32	Internal		
	toUint24	Internal		
	toUint16	Internal		
	toUint8	Internal		
	toUint256	Internal		
	toInt248	Internal		
	toInt240	Internal		
	toInt232	Internal		
	toInt224	Internal		
	toInt216	Internal		
	toInt208	Internal		
	toInt200	Internal		
	toInt192	Internal		

	toInt184	Internal		
	toInt176	Internal		
	toInt168	Internal		
	toInt160	Internal		
	toInt152	Internal		
	toInt144	Internal		
	toInt136	Internal		
	toInt128	Internal		
	toInt120	Internal		
	toInt112	Internal		
	toInt104	Internal		
	toInt96	Internal		
	toInt88	Internal		
	toInt80	Internal		
	toInt72	Internal		
	toInt64	Internal		
	toInt56	Internal		
	toInt48	Internal		
	toInt40	Internal		
	toInt32	Internal		
	toInt24	Internal		
	toInt16	Internal		
	toInt8	Internal		

	toInt256	Internal		
IERC6372	Interface			
	clock	External		-
	CLOCK_MODE	External		-
IVotes	Interface			
	getVotes	External		-
	getPastVotes	External		-
	getPastTotalSupply	External		-
	delegates	External		-
	delegate	External	✓	-
	delegateBySig	External	✓	-
IERC5805	Interface	IERC6372, IVotes		
Counters	Library			
	current	Internal		
	increment	Internal	✓	
	decrement	Internal	✓	
	reset	Internal	✓	
IERC5267	Interface			

	eip712Domain	External		-
StorageSlot	Library			
	getAddressSlot	Internal		
	getBooleanSlot	Internal		
	getBytes32Slot	Internal		
	getUint256Slot	Internal		
	getStringSlot	Internal		
	getStringSlot	Internal		
	getBytesSlot	Internal		
	getBytesSlot	Internal		
ShortStrings	Library			
	toShortString	Internal		
	toString	Internal		
	byteLength	Internal		
	toShortStringWithFallback	Internal	✓	
	toStringWithFallback	Internal		
	byteLengthWithFallback	Internal		
SignedMath	Library			
	max	Internal		
	min	Internal		

	average	Internal		
	abs	Internal		
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toString	Internal		
	toHexString	Internal		

	toHexString	Internal		
	toHexString	Internal		
	equal	Internal		
ECDSA	Library			
	_throwError	Private		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	toEthSignedMessageHash	Internal		
	toEthSignedMessageHash	Internal		
	toTypedDataHash	Internal		
	toDataWithIntendedValidatorHash	Internal		
EIP712	Implementation	IERC5267		
		Public	✓	-
	_domainSeparatorV4	Internal		
	_buildDomainSeparator	Private		
	_hashTypedDataV4	Internal		
	eip712Domain	Public		-

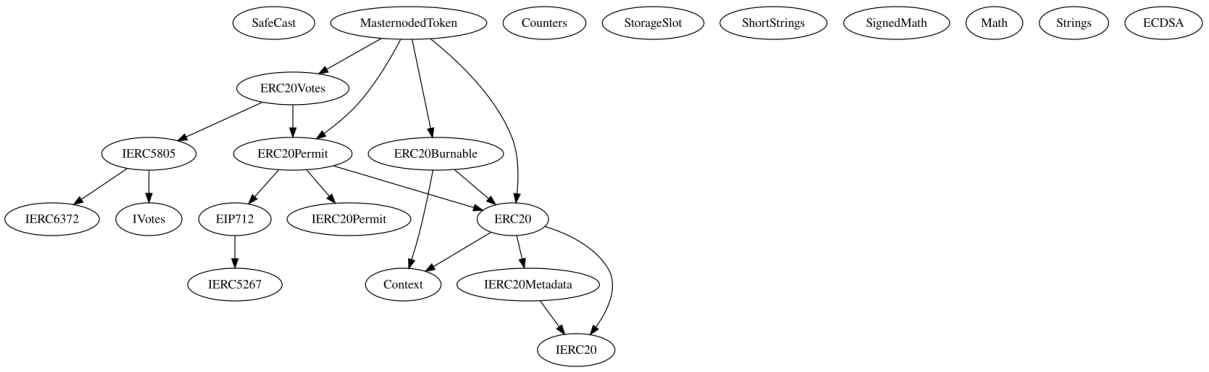
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ERC20Permit	Implementation	ERC20, IERC20Permit, EIP712		

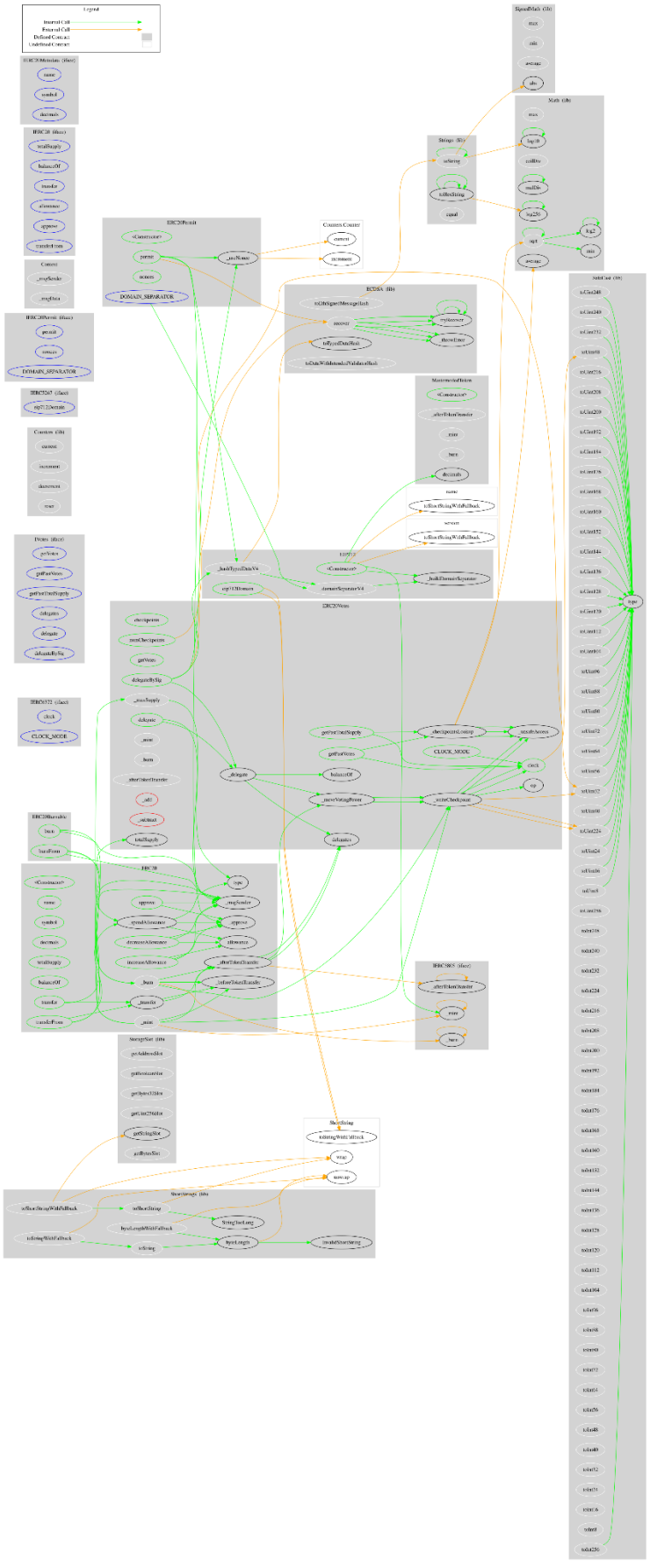
		Public	✓	EIP712
	permit	Public	✓	-
	nonces	Public		-
	DOMAIN_SEPARATOR	External		-
	_useNonce	Internal	✓	
ERC20Votes	Implementation	ERC20Permit, IERC5805		
	clock	Public		-
	CLOCK_MODE	Public		-
	checkpoints	Public		-
	numCheckpoints	Public		-
	delegates	Public		-
	getVotes	Public		-
	getPastVotes	Public		-
	getPastTotalSupply	Public		-
	_checkpointsLookup	Private		
	delegate	Public	✓	-
	delegateBySig	Public	✓	-
	_maxSupply	Internal		
	_mint	Internal	✓	
	_burn	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	_delegate	Internal	✓	

	_moveVotingPower	Private	✓	
	_writeCheckpoint	Private	✓	
	_add	Private		
	_subtract	Private		
	_unsafeAccess	Private		
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
MasternodedToken	Implementation	ERC20, ERC20Burnable, ERC20Permit, ERC20Votes		
		Public	✓	ERC20 ERC20Permit
	_afterTokenTransfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

Inheritance Graph



Flow Graph



Summary

MASTERNODED contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. MASTERNODED is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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