Cyberscope

Audit Report

MyBricks

March 2023

Type	BEP20
Network	BSC
Address	0xc2F121d1a0B2d2bEAd8f6C5d274E568b868d4913
Audited by	© cyberscope

Table of Contents

Review

Contract Name	MyRocks
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0xc2f121d1a0b2d2bead8f6c5d274e568b868d4913
Address	0xc2f121d1a0b2d2bead8f6c5d274e568b868d4913
Network	BSC
Symbol	ROCKS

Audit Updates

Initial Audit	17 Mar 2023

Source Files

Filename	SHA256
MyRocks.sol	99c42d1bb7a986ada5ee2ed58515816581b78cb312e18343a5faed3a0836fad0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CR	Code Repetition	Unresolved
●	MRS	Missing Return Statement	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CR - Code Repetition

Criticality	Minor / Informative
Location	MyRocks.sol#L1619,1624,1629
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 balance = IERC20(address(USDC)).balanceOf(address(this));
IERC20(address(USDC)).transfer(msg.sender, balance);
...
function recoverERC20A00(address _token) public onlyOperator {
    uint256 balance = IERC20(_token).balanceOf(address(this));
    IERC20(_token).transfer(msg.sender, balance);
}
```

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 `recoverERC20A00` function at the `withdrawToken` function with the proper argument, in order to avoid repeating the same code in multiple places.

MRS - Missing Return Statement

Criticality	Minor / Informative
Location	MyRocks.sol#L1616
Status	Unresolved

Description

The contract uses the `withdrawToken()` function to withdraw the balance of the USDC token to the operator's address. Based on the contract's functionality, if both `presaleOpen` and `saleOpen` are false then the contract will transfer its balance twice. In general, this might not be an issue, since the contract's balance will be zero. However, if the USDC contract's transfer function has a requirement for the amount to be greater than zero, it will revert both transactions and, as a result, the operator will not be able to withdraw the tokens.

```
function withdrawToken() public onlyOperator {
    // presale ends
    if(!presaleOpen) {
        uint256 balance = IERC20(address(USDC)).balanceOf(address(this));
        IERC20(address(USDC)).transfer(msg.sender, balance);
    }
    // public sale ends
    if(!presaleOpen && !saleOpen) {
        uint256 balance = IERC20(address(USDC)).balanceOf(address(this));
        IERC20(address(USDC)).transfer(msg.sender, balance);
    }
}
```

Recommendation

The team is advised to add a return statement at the first if-block or convert the second if-block to else-if, to ensure the contract will try to transfer only once.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	MyRocks.sol#L23
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
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MyRocks.sol#L1559,1560
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
USDC
operator
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MyRocks.sol#L1539,1549,1550,1551,1552
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 public maxPerTx = 20
address public DAO
address public TEAM
address public HOLDER
address public PRESALER
```

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	MyRocks.sol#L994,1536,1549,1550,1551,1552,1578,1597,1603,1607,1629,1635,1665,1677,1681,1685
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.

```
bytes memory _data
IERC20Ext public USDC
address public DAO
address public TEAM
address public HOLDER
address public PRESALER
address _owner
uint256 _newPrice
uint256 _newPresalePrice
bool _open

function OpenPublicSale(bool _open) public onlyOperator {
        saleOpen = _open;
    }

...
```

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	MyRocks.sol#L33,41,431,447,517,548,561,580,600,624,643,660,678,689,895,1133
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 decrement(Counter storage counter) internal {
        uint256 value = counter._value;
        require(value > 0, "Counter: decrement overflow");
        unchecked {
            counter._value = value - 1;
        }
    }

function reset(Counter storage counter) internal {
        counter._value = 0;
    }

...
```

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.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	MyRocks.sol#L1215,1217
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
bytes4 retval
bytes memory reason
```

Recommendation

By declaring local variables before using them, the contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MyRocks.sol#L1578
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.

```
address _owner
```

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	MyRocks.sol#L495,701,1223
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 {
            size := extcodesize(account)
        }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }

assembly {
                        revert(add(32, reason), mload(reason))
                    }
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MyRocks.sol#L5,48,72,97,256,332,362,395,466,714,739,766,1256,1443,1529
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;
```

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	MyRocks.sol#L1620,1625,1631,1655,1669
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(address(USDC)).transfer(msg.sender, balance)
IERC20(_token).transfer(msg.sender, balance)
USDC.transferFrom(msg.sender, address(this), _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**
Counters	Library			
	current	Internal		
	increment	Internal	✓	
	decrement	Internal	✓	
	reset	Internal	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC165	Interface			
	supportsInterface	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	getApproved	External		-
	setApprovalForAll	External	✓	-
	isApprovedForAll	External		-
	safeTransferFrom	External	✓	-

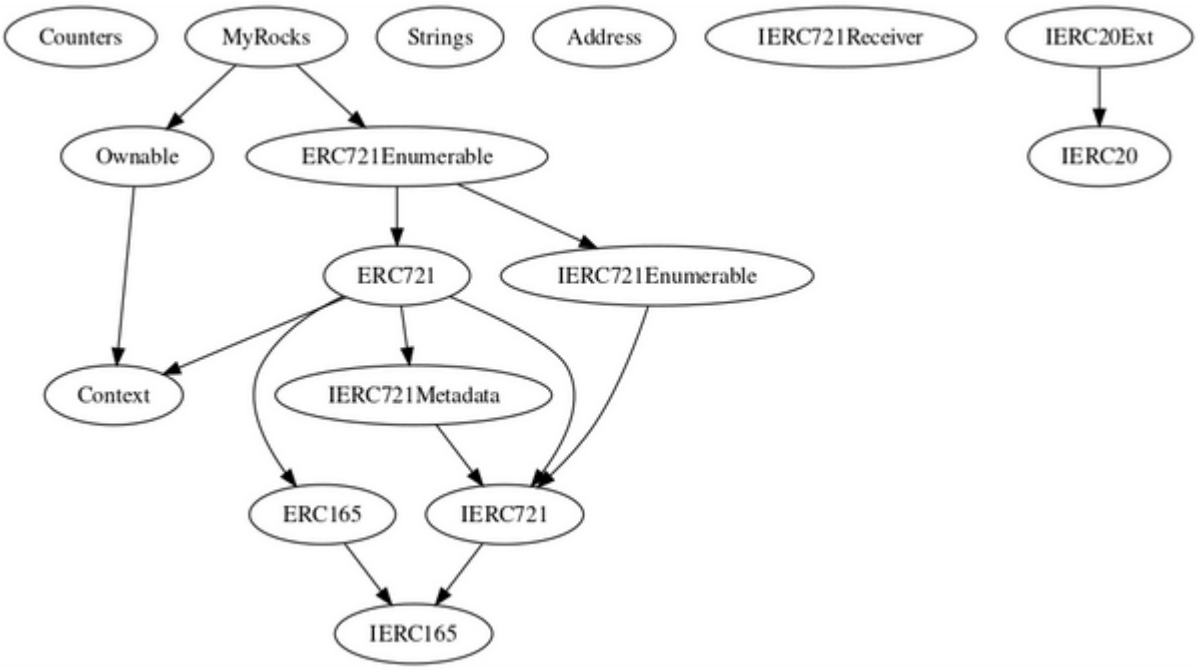
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		

	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-
	tokenURI	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-

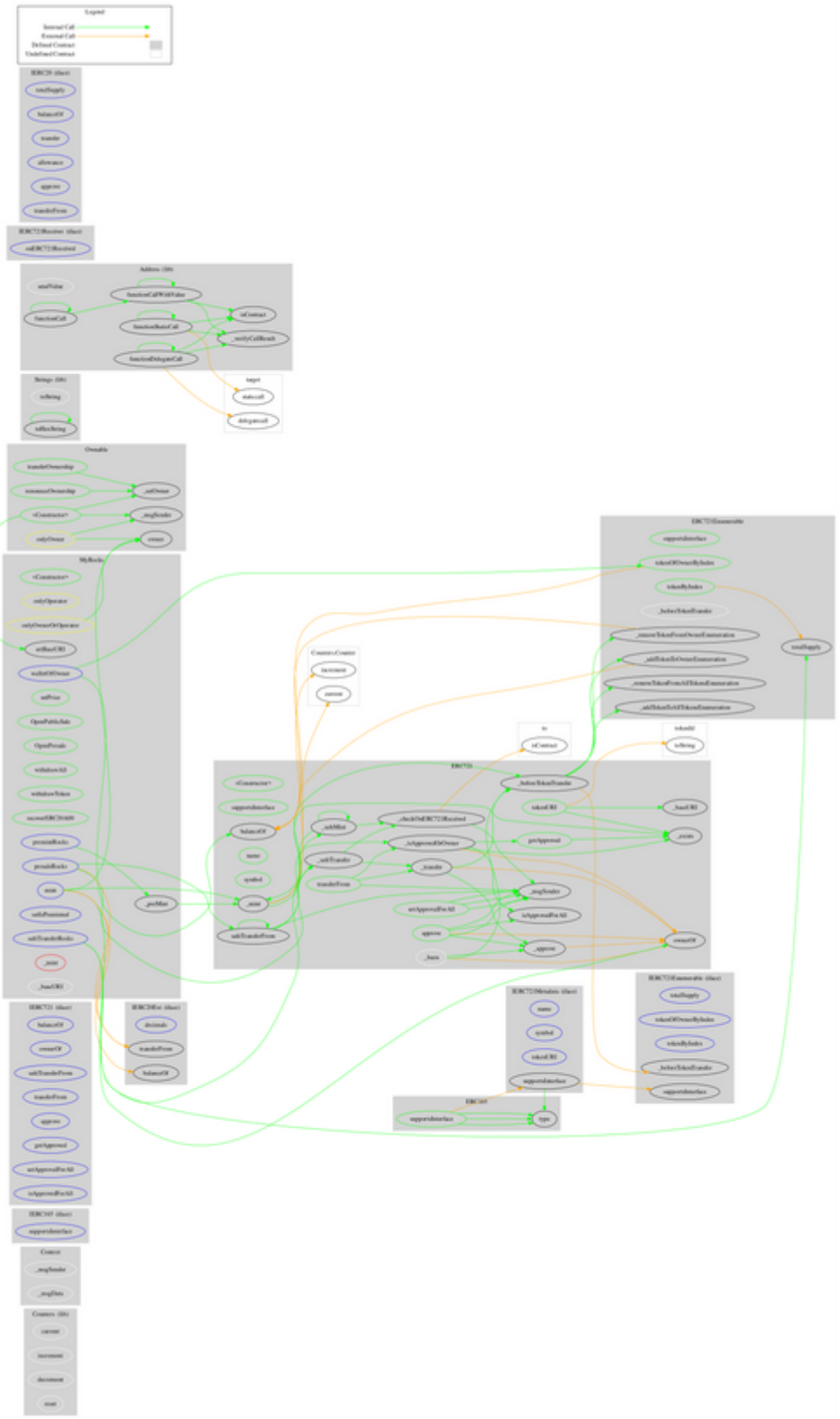
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	
	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Ext	Interface	IERC20		
	decimals	External		-
MyRocks	Implementation	ERC721Enumerable, Ownable		
		Public	✓	ERC721
	walletOfOwner	External		-
	setBaseURI	Public	✓	onlyOwnerOrOperator
	setPrice	Public	✓	onlyOwnerOrOperator
	OpenPublicSale	Public	✓	onlyOperator
	OpenPresale	Public	✓	onlyOperator
	withdrawAll	Public	✓	onlyOperator
	withdrawToken	Public	✓	onlyOperator
	recoverERC20A00	Public	✓	onlyOperator
	mint	External	✓	-
	presaleRocks	External	✓	-
	premintRocks	External	✓	onlyOwnerOrOperator
	setIsPreminted	External	✓	onlyOperator
	safeTransferRocks	External	✓	-
	_mint	Private	✓	
	_preMint	Private	✓	
	_baseURI	Internal		

Inheritance Graph



Flow Graph



Summary

MyBricks contract implements an NFT mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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