Cyberscope

Audit Report

NINJA TURTLES

May 2023

Network BSC

Address 0x183cee1544eaa6024fc3b22f0c9d94f70e251aec

Audited by © cyberscope

Table of Contents

Review

Contract Name	NINJATURTLES
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	1000 runs
Explorer	https://bscscan.com/address/0x183cee1544eaa6024fc3b22f0c9d94f70e251aec
Address	0x183cee1544eaa6024fc3b22f0c9d94f70e251aec
Network	BSC
Symbol	TURTL
Decimals	18
Total Supply	199,999,999,999

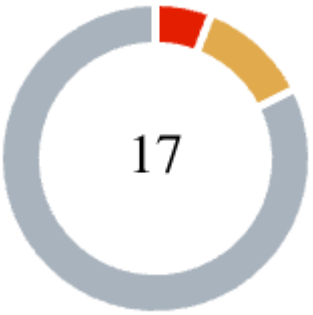
Audit Updates

Initial Audit	06 May 2023

Source Files

Filename	SHA256
NINJATURTLES.sol	fcc5f7db2ca95330ded9678f67109956d2bdd4d098695a0559f83c599bad6f7c

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	14	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

ST - Stops Transactions

Criticality	Medium
Location	NINJATURTLES.sol#L3150
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the whitelisted. Once the transactions are enabled from the owner, will not be able to be disabled again.

```
if (isWhitelisted[sender] || isWhitelisted[recipient] || inSwap) {
    super._transfer(sender, recipient, amount);
    return;
}

require(tradingEnabled,"Trading Not Started");
```

Recommendation

The contract could embody a check for not allowing setting the _maxTxAmount less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	NINJATURTLES.sol#L3185
Status	Unresolved

Description

The contract has 99% fees, as long as the sniperTaxEnabled variable is true, which is over the allowed limit of 25%.

```
if (sniperTaxEnabled) { return 9900; }
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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.

BC - Blacklists Addresses

Criticality	Medium
Location	NINJATURTLES.sol#L3316
Status	Unresolved

Description

The contract owner has the authority to massively stop addresses from transactions. The owner may take advantage of it by calling the `setIsBlacklisted` function.

```
function setIsBlacklisted(address account, bool value) external
onlyManager {
    isBlacklisted[account] = value;
    emit SetBlacklisted(account, value);
}
```

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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RMO	Redundant Mint Override	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

RMO - Redundant Mint Override

Criticality	Minor / Informative
Status	Unresolved

Description

The `_mint()` method of the `ERC20` contract is private. As a result, it cannot be called from an external user. Since the `NINJATURTLES` contract calls the `_mint()` once in the initializer method, then the `_mint()` override is redundant.

```
function _mint(address account, uint256 amount) internal virtual
override(ERC20) {
    require(!initialized, "Not Available");
    ERC20._mint(account, amount);
}
```

Recommendation

The team is advised to remove the override of the `_mint()` method since it cannot be called after the initialization.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	NINJATURTLES.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases the gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L610
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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L2150,2176,2900,3024
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.

```
used;

 public dividendsPerShareAccuracyFactor = 10 ** 36;

 private _PERMIT_TYPEHASH_DEPRECATED_SLOT;

 public maxSupply = 200000000000 * 10**18;
```

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	NINJATURTLES.sol#L6,154,155,173,1299,1300,1301,1303,1304,1305,1681,2151,2165,2212,2315,2900,2944,3031,3032,3033,3262
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);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);

 private immutable _CACHED_DOMAIN_SEPARATOR;

...

 private immutable _CACHED_THIS;


 private immutable _HASHED_NAME;


...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L2900
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
private _PERMIT_TYPEHASH_DEPRECATED_SLOT;
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L2213,2237,3268
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.

```
od = _minPeriod;


dsPerShare =
dividendsPerShare.add(dividendsPerShareAccuracyFactor.mul(amount).div(totalSha
res));


utorGas = gas;
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L620,628,653,660,668,679,689,774,792,828,839,881,892,930,943,973,999,1024,1108,1141,1154,1169,1236,1250,1435,1460,1489,1520,1530,1545,1555,1594,2010,2354,2370,2385,2394,2407
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;
    }

function max(uint256 a, uint256 b) internal pure returns (uint256)
{
        return a > b ? a : b;
    }

...
```

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	NINJATURTLES.sol#L736,739,751,755,756,757,758,759,760,766
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.

```
prod0 := div(prod0, twos)
result = prod0 * 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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L2907
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L3324,3328
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
Wallet = w;


ngWallet = w;
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	NINJATURTLES.sol#L700,1005,1116,1611
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 {
                let mm := mulmod(x, y, not(0))
                prod0 := mul(x, y)
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

mbly {
                ptr := add(buffer, add(32, length))
            }


...
```

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	NINJATURTLES.sol#L2,212,442,470,501,592,638,986,1058,1273,1379,1626,1689,1716,1801,2050,2133,2329,2447,2477,2868,2965,3004
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 0.8.17;
pragma solidity ^0.8.0;
ma solidity ^0.8.0;


solidity ^0.8.0;
...



solidity ^0.8.17;
```

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	NINJATURTLES.sol#L212,442,470,501,592,638,986,1058,1273,1379,1626,1689,1716,1801,2050,2133,2329,2447,2477,2868,2965,3004
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;

ma solidity ^0.8.0;



solidity ^0.8.0;
...
solidity ^0.8.1;

/**

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

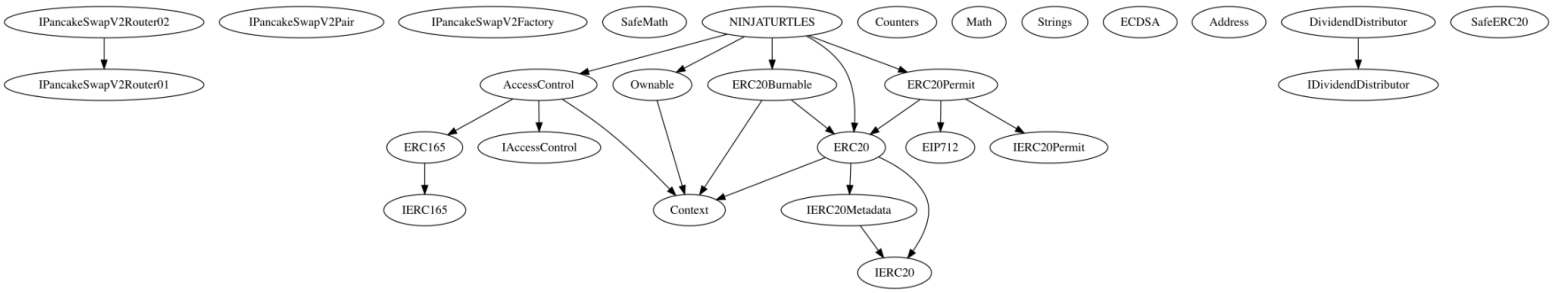
should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

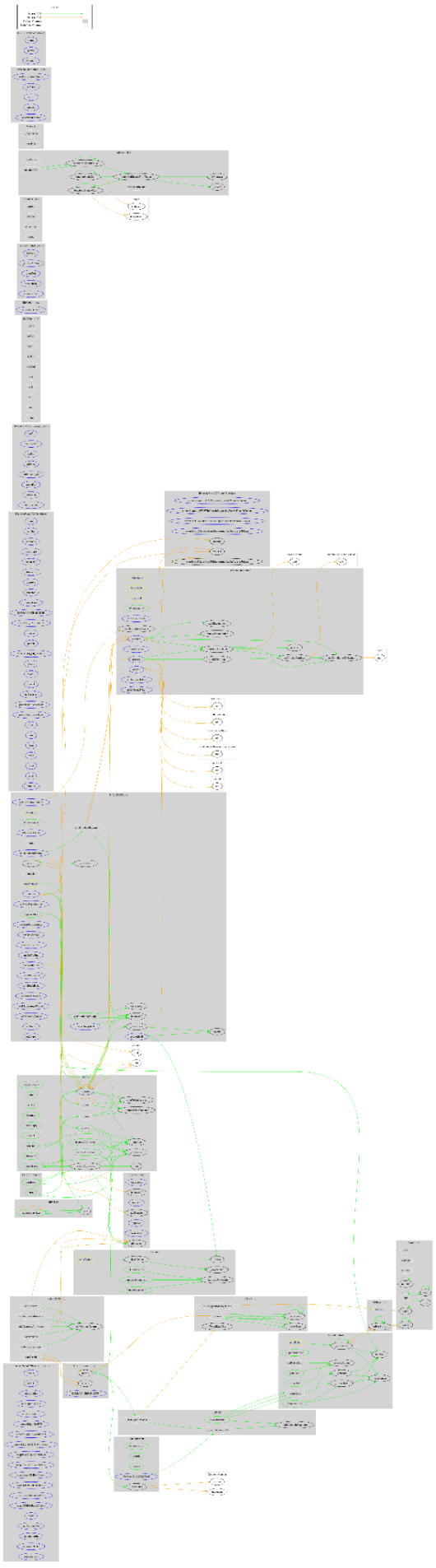
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
NINJATURTLES	Implementation	ERC20, ERC20Burnable, Ownable, AccessControl, ERC20Permit		
		Public	✓	ERC20 ERC20Permit
		External	Payable	-
	_mint	Internal	✓	
	approveMax	Public	✓	-
	initialize	External	✓	onlyOwner
	_transfer	Internal	✓	
	_takeTax	Internal	✓	
	_getTotalTax	Internal		
	_swapBack	Internal	✓	swapping
	getCirculatingSupply	Public		-
	_setIsDividendExempt	Internal	✓	
	setIsDividendExempt	External	✓	onlyManager
	setSwapBackSettings	External	✓	onlyManager
	setDistributionCriteria	External	✓	onlyManager

	setDistributorSettings	External	✓	onlyManager
	setTransferGas	External	✓	onlyManager
	removeSniperTax	External	✓	onlyManager
	enableTrading	External	✓	onlyManager
	triggerSwapBack	External	✓	onlyManager
	recoverBNB	External	✓	onlyManager
	recoverBEP20	External	✓	onlyManager
	setIsWhitelisted	External	✓	onlyManager
	setIsBlacklisted	External	✓	onlyManager
	setIsMarketMaker	External	✓	onlyManager
	setManagementWallet	External	✓	onlyManager
	setMarketingWallet	External	✓	onlyManager
	setTaxes	External	✓	onlyManager
	setShares	External	✓	onlyManager

Inheritance Graph



Flow Graph



Summary

NINJA TURTLES 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 transaction, manipulate the fees and blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks.

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