Cyberscope

Audit Report

Circle Launchpad Locker

December 2022

Github	https://github.com/monkey-shanti/Circle-Launchpad
Commit	831864399fdc88aaf191f8594ca0d22d09080652
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Contract Review 3
Audit Updates 3
Source Files 3
Introduction 4
Roles 4
Owner Role 4
Locker Owner Role 4
User Role 4
Contract Diagnostics 5
ITGC - ID Token Generation Conflict 6
Description 6
Recommendation 6
NALTD - Normal and LP Token Duplication 7
Description 7
Recommendation 7
LTM - LP Token Mocking 8
Description 8
TPC - Transposition Property Check 9
Description 9
Recommendation 9
PI - Performance Improvement 10
Description 10
Recommendation 11
LLS - Locker Logic Simplification 12
Description 12
Recommendation 12
L04 - Conformance to Solidity Naming Conventions 13
Description 13
Recommendation 14
L12 - Using Variables before Declaration 15

Description 15
Recommendation 15
L13 - Divide before Multiply Operation 16
Description 16
Recommendation 16
L14 - Uninitialized Variables in Local Scope 17
Description 17
Recommendation 17
L16 - Validate Variable Setters 18
Description 18
Recommendation 18
L18 - Multiple Pragma Directives 19
Description 19
Recommendation 19
L19 - Stable Compiler Version 20
Description 20
Recommendation 20
Contract Functions 21
Contract Flow 26
Inheritance Graph 27
Summary 28
Disclaimer 29
About Cyberscope 30

Contract Review

Contract Name	CircleLocker
Testing Deploy	https://testnet.bscscan.com/address/0x475c78eEbA264c624e31db70C29F1bA6141BF769

Audit Updates

Initial Audit	20 Dec 2022

Source Files

Filename	SHA256
interfaces/IUniswapV2Pair.sol	123cb0b5508420d58ba182bc3218fb9c670efd16de72f33415b0657a93873538
interfaces/PoolLibrary.sol	2d5a552523e29e49a13d68a4c7d81b99541e958b179d5128ab5baaa14bdcda54
Locker.sol	1e99393d5c7267812204bc2215fb62377c458a150e8cb0810a3c27d60f17fa00

Introduction

The Locker contract implements a locker mechanism.

Roles

The contract has three roles.

Owner Role

The owner role has the authority to WithdrawBNB.

Locker Owner Role

- editLockDescription
- editLock
- withdrawableTokens
- unlock
- transferLockOwnership

User Role

The users have the authority to

- multipleVestingLock
- vestingLock
- lock

Contract Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ITGC	ID Token Generation Conflict	Unresolved
●	NALTD	Normal and LP Token Duplication	Unresolved
●	LTM	LP Token Mocking	Unresolved
●	TPC	Transposition Property Check	Unresolved
●	PI	Performance Improvement	Unresolved
●	LLS	Locker Logic Simplification	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

ITGC - ID Token Generation Conflict

Criticality	Minor / Informative
Location	Locker.sol#L414
Status	Unresolved

Description

The contract generates ids based on the fact that the previous versions will not reach the ID_PADDING amount of locks. This is an arbitrary assumption that may break if someone abuses the previous versions and generates an ID_PADDING amount of locks.

```
id = _locks.length + ID_PADDING;
```

Recommendation

The contract could use a reverse counter mechanism so it will be essentially impossible to generate such a large amount of lockers.

```
const id = ~uint256(0) - _locks.length
```

```
_getActualIndex -> ~uint256(0) - id
```

NALTD - Normal and LP Token Duplication

Criticality	Minor / Informative
Location	Locker.sol#L394,361
Status	Unresolved

Description

The user has the ability to lock the LP token in the normal token lock. Later, if the user locks the LP token the factory address will not be registered since the token will already exist. Hence, since the contract uses the `bool isLpToken = tokenInfo.factory != address(0);` to determine the locker state, the contract will always mark it as a simple token and the LP token locker will be inaccessible.

```
CumulativeLockInfo storage tokenInfo = cumulativeLockInfo[token];
if (tokenInfo.token == address(0)) {
    tokenInfo.token = token;
    tokenInfo.factory = address(0);
}

CumulativeLockInfo storage tokenInfo = cumulativeLockInfo[token];
if (tokenInfo.token == address(0)) {
    tokenInfo.token = token;
    tokenInfo.factory = factory;
}
```

Recommendation

The contract could implement a structure that is not based on the fact that a token address will be either a normal token or an LP token. Alternatively, the contract could check the factory shape in order to not allow locking an LP token as a simple token.

LTM - LP Token Mocking

Criticality	Minor / Informative
Location	Locker.sol#L908
Status	Unresolved

Description

The user can create a contract that implements the `factory()`, `getPair()`, and `token0()`, `token1()` methods in order to mock the LP token validator. As a result, the user will be able to lock an LP Token that essentially is not an LP token.

```
function _isValidLpToken(address token, address factory)
private
view
returns (bool)
{
    IUniswapV2Pair pair = IUniswapV2Pair(token);
    address factoryPair = IUniswapV2Factory(factory).getPair(
        pair.token0(),
        pair.token1()
    );
```

TPC - Transposition Property Check

Criticality	Minor / Informative
Location	Locker.sol#L183
Status	Unresolved

Description

According to the transposition property, if the sum of two integers is less than n, then each integer will be less than n.

```
require(tgeBps > 0 && tgeBps < 10_000, "Invalid bips for TGE");
require(cycleBps > 0 && cycleBps < 10_000, "Invalid bips for cycle");
require(
    tgeBps + cycleBps <= 10_000,
    "Sum of TGE bps and cycle should be less than 10000"
);
```

Recommendation

The contract could skip the individual checks since the sum is checked.

PI - Performance Improvement

Criticality	Minor / Informative
Location	Locker.sol#L251,566
Status	Unresolved

Description

Since the owners and the amount array have the same size and the owners' array is iterated, then the initial _sumAmount() calculation is redundant. The calculation of the amount could be moved inside the owners' loop.

```
uint256 sumAmount = _sumAmount(amounts);
uint256 count = owners.length;
uint256[] memory ids = new uint256[](count);
for (uint256 i = 0; i < count; i++) {
    ids[i] = _createLock(
        owners[i],
        token,
        isLpToken,
        amounts[i],
        vestingSettings[0], // TGE date
        vestingSettings[1], // TGE bps
        vestingSettings[2], // cycle
        vestingSettings[3], // cycle bps
        description
    );
    emit LockAdded(
        ids[i],
        token,
        owners[i],
        amounts[i],
        vestingSettings[0] // TGE date
    );
}
_safeTransferFromEnsureExactAmount(
    token,
    msg.sender,
    address(this),
    sumAmount
);
```

Since the `block.timestamp < userLock.tgeDate` is checked in the beginning of the method, then the expression `if (block.timestamp >= userLock.tgeDate) {` is redundant.

```
if (block.timestamp < userLock.tgeDate) return 0;
if (userLock.cycle == 0) return 0;

uint256 tgeReleaseAmount = Math.mulDiv(
    userLock.amount,
    userLock.tgeBps,
    10_000
);
uint256 cycleReleaseAmount = Math.mulDiv(
    userLock.amount,
    userLock.cycleBps,
    10_000
);
uint256 currentTotal = 0;
if (block.timestamp >= userLock.tgeDate) {
    currentTotal =
    (((block.timestamp - userLock.tgeDate) / userLock.cycle) *
    cycleReleaseAmount) +
    tgeReleaseAmount; // Truncation is expected here
}
```

Recommendation

The team is advised to take into consideration these segment and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

- The contract could merge the two loops in one and calculate the sumAmount in the owners loop.
- The contract could remove the if branch since the expression will always yield positive value.

LLS - Locker Logic Simplification

Criticality	Minor / Informative
Location	Locker.sol#L24
Status	Unresolved

Description

The normal and vesting lock could be the same since the normal lock is equal to the vesting lock with 100% return in the first circle.

```
Normal lock == Vesting with:
tgeBps = 10_000
tgeDate = lock date
cycleBps = lock start date
cycle = 1
```

Recommendation

The contract could merge and simplify the logic of the vesting and normal token lock.

```
struct Lock {
    uint256 id;
    address token;
    address owner;
    uint256 amount;
    uint256 lockDate;
    uint256 tgeDate; // TGE date for vesting locks, unlock date for
normal locks
    uint256 tgeBps; // In bips. Is 0 for normal locks
    uint256 cycle; // Is 0 for normal locks
    uint256 cycleBps; // In bips. Is 0 for normal locks
    uint256 unlockedAmount;
    string description;
    bool isVesting;
}
```

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Locker.sol#L921
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 your 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 your code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WithdrawBNB(address payable _reciever, uint256 _amount)
    public
    onlyOwner
    {
        _reciever.transfer(_amount);
    }
address payable _reciever
uint256 _amount
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

You can find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	contracts/Locker.sol#L895
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.

```
address factory
```

Recommendation

By declaring local variables before using them, you can ensure that your contract 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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/Locker.sol#L581
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 a loss of prediction.

```
currentTotal =
            (((block.timestamp - userLock.tgeDate) / userLock.cycle) *
            cycleReleaseAmount) +
            tgeReleaseAmount
```

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	contracts/Locker.sol#L894,895
Status	Unresolved

Description

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

```
address possibleFactoryAddress
address factory
```

Recommendation

By initializing local variables before using them, you can help ensure that your contract functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/Locker.sol#L925
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.

```
_reciever.transfer(_amount)
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/Locker.sol#L2
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 ^0.8.4;
```

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, you can avoid conflicts and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/Locker.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 you to specify a minimum version of the Solidity compiler that must be used to compile your contract code. This is useful because it allows you to ensure that your contract will be compiled using a version of the compiler that is known to be compatible with your code.

```
pragma solidity ^0.8.4;
```

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.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-

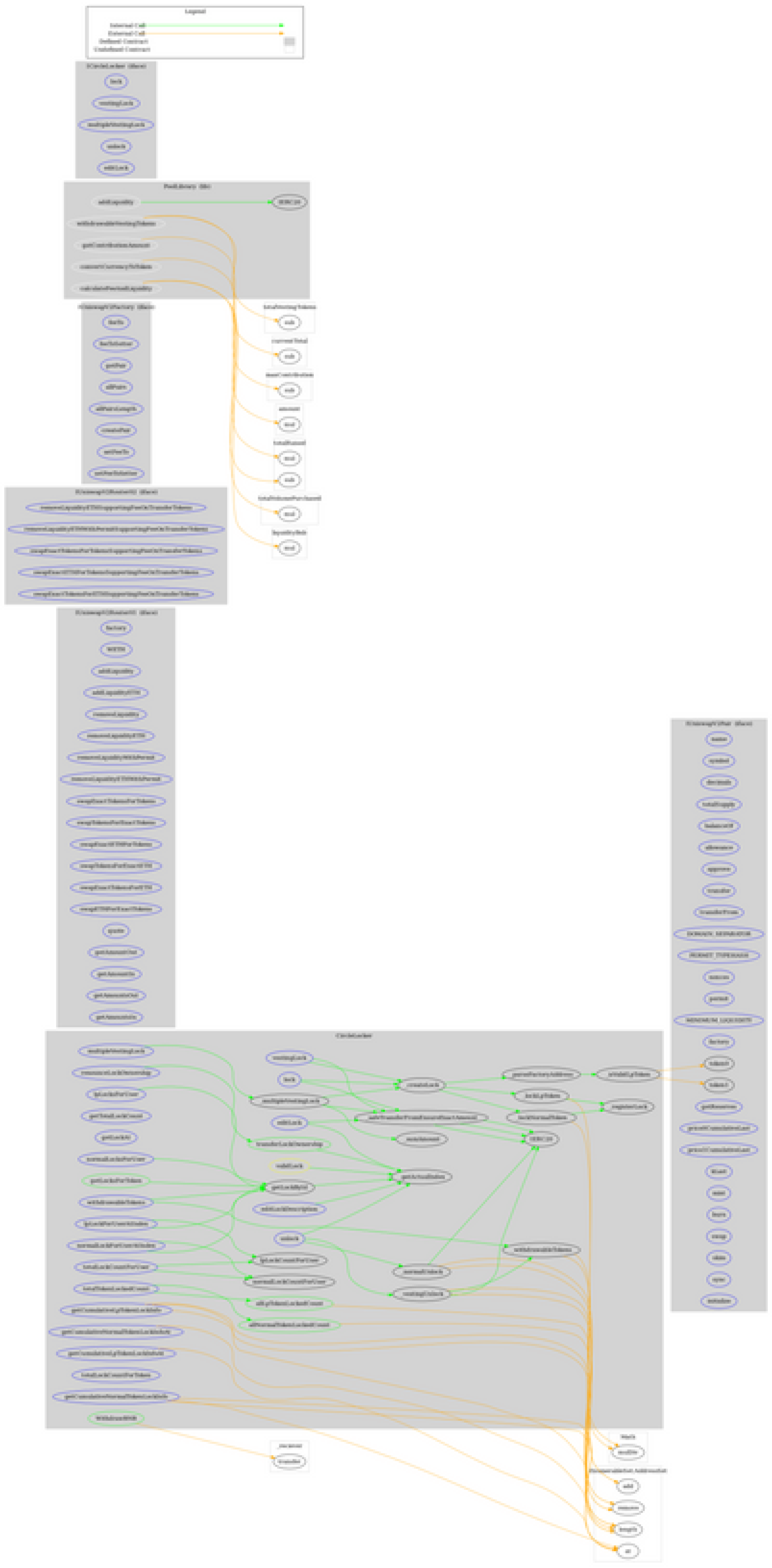
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		

	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
PoolLibrary	Library			
	withdrawableVestingTokens	Internal		
	getContributionAmount	Internal		
	convertCurrencyToToken	Internal		
	addLiquidity	Internal	✓	
	calculateFeeAndLiquidity	Internal		
ICircleLocker	Interface			
	lock	External	✓	-
	vestingLock	External	✓	-
	multipleVestingLock	External	✓	-

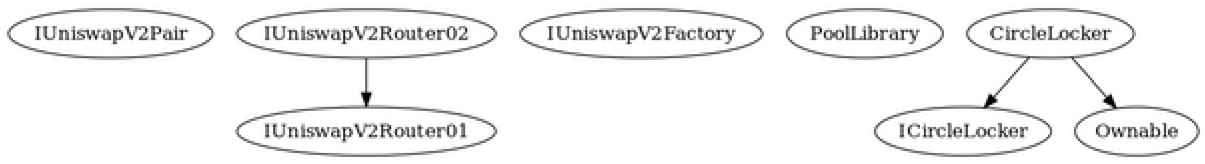
	unlock	External	✓	-
	editLock	External	✓	-
CircleLocker	Implementation	ICircleLocker, Ownable		
	lock	External	✓	-
	vestingLock	External	✓	-
	multipleVestingLock	External	✓	-
	_multipleVestingLock	Internal	✓	
	_sumAmount	Internal		
	_createLock	Internal	✓	
	_lockLpToken	Private	✓	
	_lockNormalToken	Private	✓	
	_registerLock	Private	✓	
	unlock	External	✓	validLock
	_normalUnlock	Internal	✓	
	_vestingUnlock	Internal	✓	
	withdrawableTokens	External		-
	_withdrawableTokens	Internal		
	editLock	External	✓	validLock
	editLockDescription	External	✓	validLock
	transferLockOwnership	Public	✓	validLock
	renounceLockOwnership	External	✓	-
	_safeTransferFromEnsureExactAmount	Internal	✓	
	getTotalLockCount	External		-
	getLockAt	External		-
	getLockById	Public		-
	allLpTokenLockedCount	Public		-
	allNormalTokenLockedCount	Public		-
	getCumulativeLpTokenLockInfoAt	External		-

	getCumulativeNormalTokenLockInfoAt	External		-
	getCumulativeLpTokenLockInfo	External		-
	getCumulativeNormalTokenLockInfo	External		-
	totalTokenLockedCount	External		-
	lpLockCountForUser	Public		-
	lpLocksForUser	External		-
	lpLockForUserAtIndex	External		-
	normalLockCountForUser	Public		-
	normalLocksForUser	External		-
	normalLockForUserAtIndex	External		-
	totalLockCountForUser	External		-
	totalLockCountForToken	External		-
	getLocksForToken	Public		-
	_getActualIndex	Internal		
	_parseFactoryAddress	Internal		
	_isValidLpToken	Private		
	WithdrawBNB	Public	✓	onlyOwner

Contract Flow



Inheritance Graph



Summary

The Locker contract implements a locker 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