Cyberscope

Audit Report

Xpat Ink

February 2023

Type	BEP20
Network	BSC
Address	0xC1f7F6D6338707A362f35289D882f12F364c2785
Audited by	© cyberscope

Table of Contents

Review

Contract Name	XPatInk
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0xc1f7f6d6338707a362f35289d882f12f364c2785
Address	0xc1f7f6d6338707a362f35289d882f12f364c2785
Network	BSC
Symbol	XINK
Decimals	18
Total Supply	24,000,000

Audit Updates

Initial Audit	25 Feb 2023 https://github.com/cyberscope-io/audits/tree/main/xink/v1/audit.pdf
Corrected Phase 2	28 Feb 2023

Source Files

Filename	SHA256
contracts/AntiSniper.sol	1569c52e03a8ff118941ad557fa59cdb9acf76ba7d270305e21409942dd4d27f
contracts/BaseErc20.sol	bfcfefef8535cc300412ec88664418b903005136188c44ea18c0290f3df6b517
contracts/Burnable.sol	ec14be12f21af244699151eccfed96ad5a286a7508ca4cf14f7276da50d4b791
contracts/Interfaces.sol	a022a8f0668797c9010037a02f15e4e1cf54b107ef4cf68f236898a93161d4bb
contracts/Libraries.sol	5d114180c3e047b657f6c2befd65c1b21ce0e9ad160c31988426457d92a2f698
contracts/Taxable.sol	d923d19600e034158c742693ad061d010f25f95334a68d59654b0087a915e8b3
contracts/TaxDistributor.sol	3aac56d4190c883c685d2334226071f2c86a74e90c0d660b34aa7556a4b1cc29
contracts/Xpatink.sol	6d525e10782c6399cfac26da2d357602f9f4cc0171ade2a6c65b5b2b29d1d1a9

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RS	Redundant Statements	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	USV	Unused State Variables	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RS - Redundant Statements

Criticality	Minor / Informative
Location	contracts/TaxDistributor.sol#L101
Status	Unresolved

Description

The `checkTokenAmount()` method is called twice but will always yield the same result between the two sequential calls. As a result the condition `checkTokenAmount(token, totalTokens) != totalTokens` will equal false. So the if branch is redundant.

```
totalTokens = checkTokenAmount(token, totalTokens);
if (checkTokenAmount(token, totalTokens) != totalTokens) {
    emit DistributionError("Insufficient tokens to swap. Please add more
tokens");
    return;
}
```

Recommendation

The team is advised to remove the entire code segment since it always results in a dead execution context.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/Taxable.sol#L103
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `setTaxDistributionThresholds` 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.

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 setTaxDistributionThresholds(uint256 minAmount, uint256 minTime)
external onlyOwner {
    minimumTokensBeforeSwap = minAmount;
    minimumTimeBetweenSwaps = minTime;
}
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

USV - Unused State Variables

Criticality	Minor / Informative
Status	Unresolved

Description

There are state variables that are declared but not used by the contract. As a result, they increase gas consumption and decrease code readability.

```
mapping (address => bool) public excludedFromSelling;
require(excludedFromSelling[from] == false, "address is not allowed to sell");

...

bool public isTradingEnabled = true;
require((isTradingEnabled && launched)...

...

function createDistributorTax(string memory name, uint256 buyTax, uint256
sellTax, address wallet, bool convertToNative) external override onlyToken {
    taxes.push(Tax(name, buyTax, sellTax, 0, TaxType.DISTRIBUTOR, wallet, 0,
convertToNative));
}

function createDividendTax(string memory name, uint256 buyTax, uint256
sellTax, address dividendDistributor, bool convertToNative) external override
onlyToken {
    taxes.push(Tax(name, buyTax, sellTax, 0, TaxType.DIVIDEND,
dividendDistributor, 0, convertToNative));
}

function createBurnTax(string memory name, uint256 buyTax, uint256 sellTax)
external override onlyToken {
    taxes.push(Tax(name, buyTax, sellTax, 0, TaxType.BURN, address(0), 0,
false));
}

function createLiquidityTax(string memory name, uint256 buyTax, uint256
sellTax, address holder) external override onlyToken {
    taxes.push(Tax(name, buyTax, sellTax, 0, TaxType.LIQUIDITY, holder, 0,
false));
}
...
bool internal _useSafeTransfer;
if (_useSafeTransfer) {...
```

Recommendation

The team is advised to remove these variables from the source code since they are redundant.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/TaxDistributor.sol#L57
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`.

```
_route
```

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	contracts/BaseErc20.sol#L13,20
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.

```
bool internal _useSafeTransfer
bool public isTradingEnabled = true
```

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/Xpatink.sol#L69 contracts/Taxable.sol#L21 contracts/BaseErc20.sol#L10,11,12,13,44,61,71 contracts/AntiSniper.sol#L38
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 _owner
mapping (address => uint256) internal _balances
mapping (address => mapping (address => uint256)) internal _allowed
uint256 internal _totalSupply
bool internal _useSafeTransfer
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	contracts/BaseErc20.sol#L174
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
owner = who
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/Xpatink.sol#L97
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.

```
zkTokenThreshold = amount
```

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	contracts/AntiSniper.sol#L38,49,91
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 configure(address _owner) internal virtual override {
        isNeverSniper[_owner] = true;
        super.configure(_owner);
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/TaxDistributor.sol#L45 contracts/Taxable.sol#L31,53,99 contracts/BaseErc20.sol#L219 contracts/AntiSniper.sol#L50,52,58,75,78,94
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(inSwap == false, "already swapping")
excludedFromTax[from] == false && excludedFromTax[to] == false && launched

launched &&
            autoSwapTax &&
            exchanges[to] &&
            swapStartTime + 60 <= block.timestamp &&
            timeSinceLastSwap >= minimumTimeBetweenSwaps &&
            _balances[address(taxDistributor)] >= minimumTokensBeforeSwap &&
            taxDistributor.inSwap() == false
require(exchanges[who] == false || enabled == false, "Cannot exclude an
exchange from tax")
require(excludedFromSelling[from] == false, "address is not allowed to sell")
require(enableSniperBlocking == false || isSniper[msg.sender] == false,
"sniper rejected")
launched && from != owner && isNeverSniper[from] == false && isNeverSniper[to]
== false
mhPercentage > 0 && exchanges[to] == false

...
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/TaxDistributor.sol#L92,208,219,231
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.

```
uint256 totalTokens
bool updated
```

Recommendation

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/Xpatink.sol#L3
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 ^0.8.7;
```

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/TaxDistributor.sol#L147
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.

```
token.transfer(taxes[i].location, checkTokenAmount(token, taxes[i].taxPool))
```

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

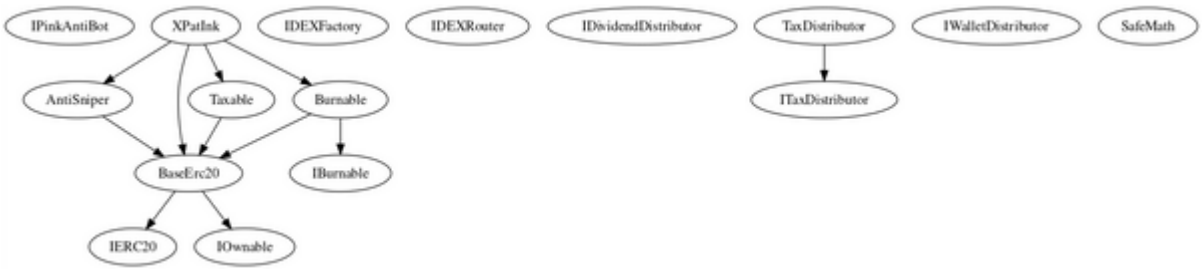
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IPinkAntiBot	Interface			
	setTokenOwner	External	✓	-
	onPreTransferCheck	External	✓	-
AntiSniper	Implementation	BaseErc20		
	configure	Internal	✓	
	launch	Public	✓	onlyOwner
	preTransfer	Internal	✓	
	calculateTransferAmount	Internal	✓	
	mhAmount	Public		-
	msAmount	Public		-
	sniperTax	Public		-
	setSniperBlocking	External	✓	onlyOwner
	setBlockLogProtection	External	✓	onlyOwner
	setHighTaxCountdown	External	✓	onlyOwner
	setPinkAntiBot	External	✓	onlyOwner
	setMsPercentage	External	✓	onlyOwner
	setMhPercentage	External	✓	onlyOwner
BaseErc20	Implementation	IERC20, IOwnable		
	configure	Internal	✓	
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-

	transfer	External	✓	tradingEnabled
	approve	External	✓	tradingEnabled
	transferFrom	External	✓	tradingEnabled
	increaseAllowance	External	✓	tradingEnabled
	decreaseAllowance	External	✓	tradingEnabled
	launch	Public	✓	onlyOwner
	preTransfer	Internal	✓	
	calculateTransferAmount	Internal	✓	
	postTransfer	Internal	✓	
	changeOwner	External	✓	onlyOwner
	removeBnb	External	✓	onlyOwner
	transferTokens	External	✓	onlyOwner
	setCanAlwaysTrade	External	✓	onlyOwner
	setExchange	External	✓	onlyOwner
	getRouterAddress	Internal		
	_transfer	Private	✓	
Taxable	Implementation	BaseErc20		
	configure	Internal	✓	
	calculateTransferAmount	Internal	✓	
	preTransfer	Internal	✓	
	sellTax	External		-
	buyTax	External		-
	taxDistributorAddress	External		-
	setAutoSwaptax	External	✓	onlyOwner
	setExcludedFromTax	External	✓	onlyOwner
	setTaxDistributionThresholds	External	✓	onlyOwner
	runSwapManually	External	✓	isLaunched

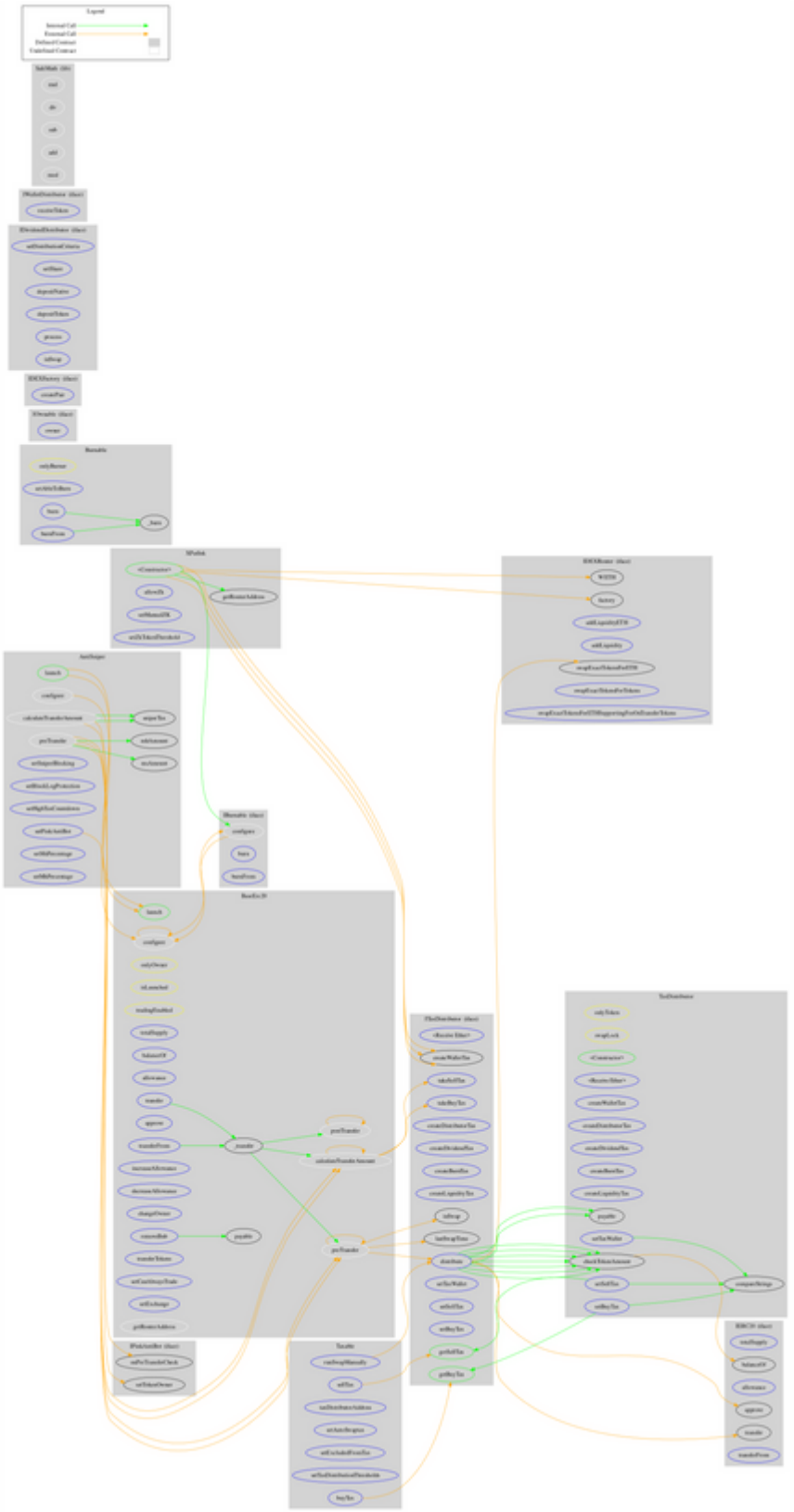
TaxDistributor	Implementation	ITaxDistributor		
		Public	✓	-
		External	Payable	-
	createWalletTax	External	✓	onlyToken
	createDistributorTax	External	✓	onlyToken
	createDividendTax	External	✓	onlyToken
	createBurnTax	External	✓	onlyToken
	createLiquidityTax	External	✓	onlyToken
	distribute	External	Payable	onlyToken swapLock
	getSellTax	Public		onlyToken
	getBuyTax	Public		onlyToken
	setTaxWallet	External	✓	onlyToken
	setSellTax	External	✓	onlyToken
	setBuyTax	External	✓	onlyToken
	takeSellTax	External	✓	onlyToken
	takeBuyTax	External	✓	onlyToken
	compareStrings	Private		
	checkTokenAmount	Private		
XPatInk	Implementation	BaseErc20, AntiSniper, Burnable, Taxable		
		Public	✓	-
	launch	Public	✓	onlyOwner
	configure	Internal	✓	
	preTransfer	Internal	✓	
	calculateTransferAmount	Internal	✓	
	postTransfer	Internal	✓	

	allowZk	External		-
	setManualZK	External	✓	onlyOwner
	setZkTokenThreshold	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Xpat Ink contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Xpat Ink is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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. The fees are fixed to 5%.

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