Cyberscope

Audit Report

ØxLiquidity

February 2024

Network	ETH
Address	0xd377F28245BC505190c8f34D2bFE5f215754f634
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
●	FSA	Fixed Swap Address	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L08	Tautology or Contradiction	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ZeroXLiquidity
Compiler Version	v0.8.23+commit.f704f362
Optimization	200 runs
Explorer	https://etherscan.io/address/0xd377f28245bc505190c8f34d2bfe5f215754f634
Address	0xd377f28245bc505190c8f34d2bfe5f215754f634
Network	ETH
Symbol	0xLP
Decimals	9
Total Supply	100,000,000

Audit Updates

Initial Audit	04 Feb 2024

Source Files

Filename	SHA256
ZeroXLiquidity.sol	f7480aa374db7f0af0c61916c9253c87dcd0f619af487f1f84c1a887d1631e95

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	14

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

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L220
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(routerAddress);
    uniswapV2Router = _uniswapV2Router;
    uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L222
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
uniswapV2Pair
```

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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L410,416
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_msgSender() == _devAddress)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L181,216,248
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

Specifically, reflection-related variables are assigned to `_tOwned`, which is not intended to be used for reflection.

```
mapping(address => uint256) private _tOwned;

_tOwned[_msgSender()] = _rTotal;

function balanceOf(address account) public view override
returns (uint256) {
    return tokenFromRef(_tOwned[account]);
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L392,449,546
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmount,
    0,
    path,
    address(this),
    block.timestamp
);

function _takeCom(uint256 tCom) private {
    uint256 currentRate = _getRate();
    uint256 rCom = tCom.mul(currentRate);
    _tOwned[address(this)] = _tOwned[address(this)].add(rCom);
}

function excludeAccountsFromFees(address[] calldata accounts, bool
excluded) public onlyOwner {
    for(uint256 i = 0; i < accounts.length; i++) {
        _isExcludedFromFee[accounts[i]] = excluded;
    }
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L93
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#92
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 to 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 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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L186,187
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 private _rTotal = (MAX - (MAX % _tTotal))
uint256 private _tFeeTotal
```

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	ZeroXLiquidity.sol#L154,177,178,179,185,204,205,206,405,534
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);
string private constant _name = "0xLiquidity"
string private constant _symbol = "0xLP"
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 100_000_000 * 10 **
_decimals
uint256 public _maxTxAmount = _tTotal.div(100)
uint256 public _maxWalletSize = _tTotal.div(100)
uint256 public _swapTokensAtAmount = 1000 * 10**_decimals
bool _tradingOpen
bool _swapEnabled
```

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	ZeroXLiquidity.sol#L187
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.

```
uint256 private _tFeeTotal
```

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	ZeroXLiquidity.sol#L525,531,539,543
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.

```
_taxFeeOnBuy = taxFeeOnBuy
_swapTokensAtAmount = swapTokensAtAmount
_maxTxAmount = maxTxAmount
_maxWalletSize = maxWalletSize
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L522,523
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(taxFeeOnBuy >= 0 && taxFeeOnBuy <= 95, "Buy tax must be between 0% and 95%")
require(taxFeeOnSell >= 0 && taxFeeOnSell <= 95, "Sell tax must be between 0% and 95%")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L66,218
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.

```
_owner = msgSender
_devAddress = payable(devAddress)
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ZeroXLiquidity.sol#L23
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 ^0.8.23;
```

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

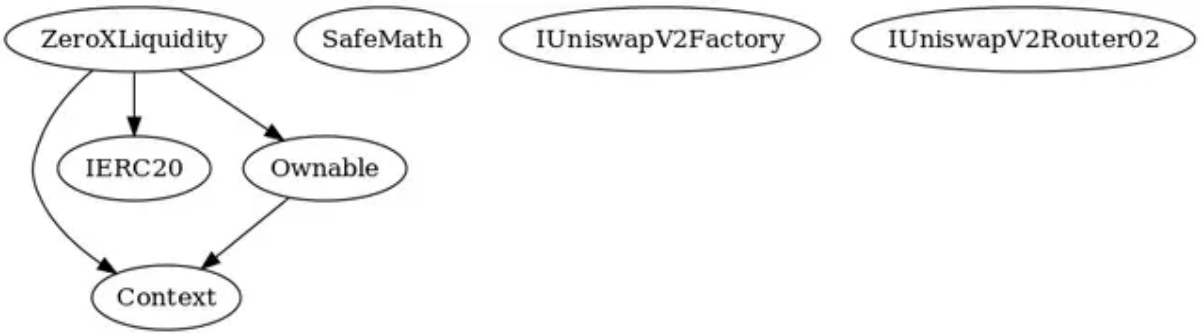
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMath	Library			
	add	Internal		

	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
ZeroXLiquidity	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-

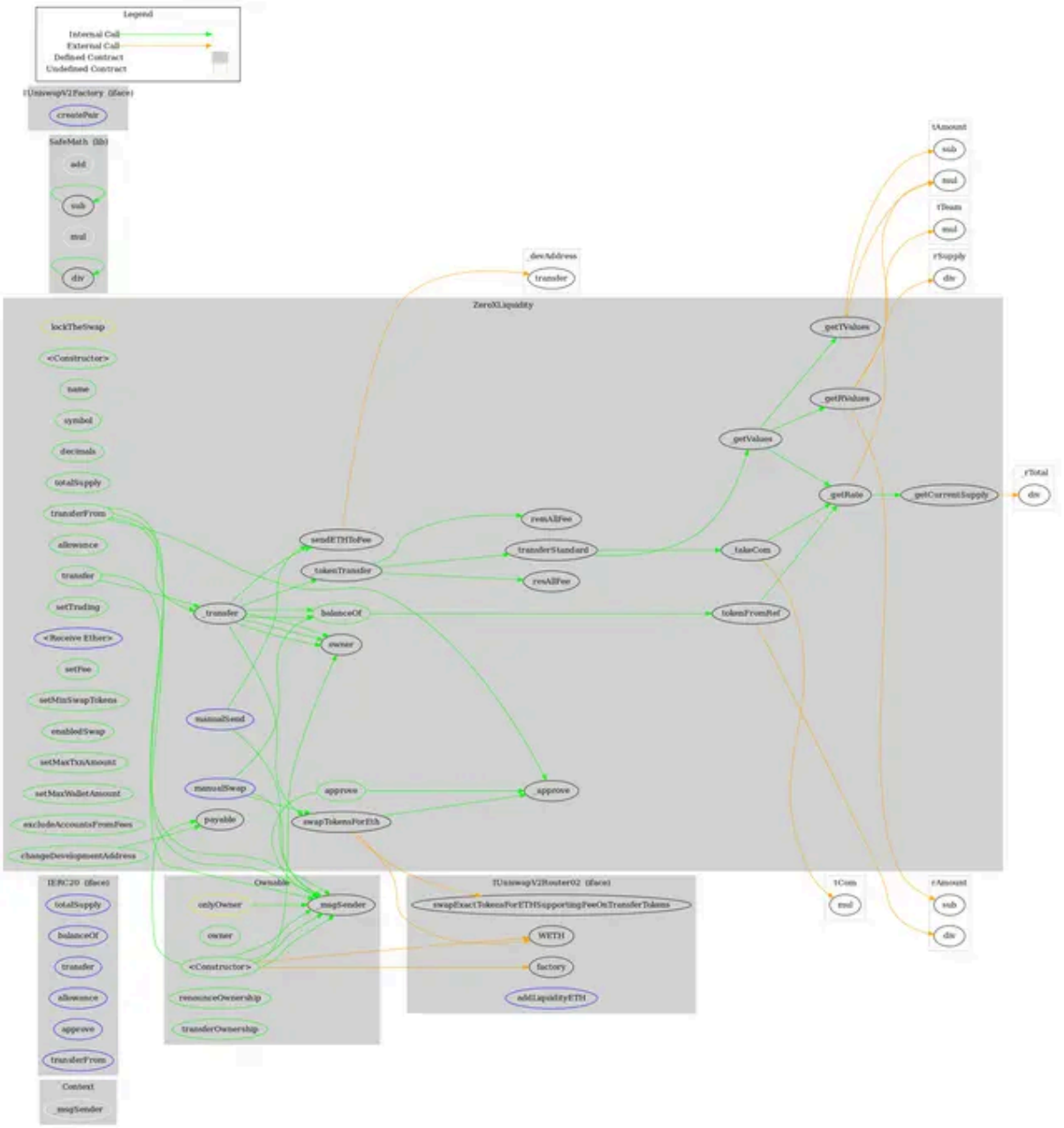
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	tokenFromRef	Private		
	remAllFee	Private	✓	
	resAllFee	Private	✓	
	_approve	Private	✓	
	_transfer	Private	✓	
	swapTokensForEth	Private	✓	lockTheSwap
	sendETHToFee	Private	✓	
	setTrading	Public	✓	onlyOwner
	manualSwap	External	✓	-
	manualSend	External	✓	-
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_takeCom	Private	✓	
		External	Payable	-
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		

	setFee	Public	✓	onlyOwner
	setMinSwapTokens	Public	✓	onlyOwner
	enabledSwap	Public	✓	onlyOwner
	setMaxTxnAmount	Public	✓	onlyOwner
	setMaxWalletAmount	Public	✓	onlyOwner
	excludeAccountsFromFees	Public	✓	onlyOwner
	changeDevelopmentAddress	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
SafeMath (lib)
add
sub
mul
div
devAddress
transfer
ZeroXLiquidity
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
transferFrom
allowance
transfer
setTrading
<Receive Ether>
setFee
setMinSwapTokens
enabledSwap
setMaxTxnAmount
setMaxWalletAmount
excludeAccountsFromFees
changeDevelopmentAddress
sendETHToFee
tokenTransfer
_transfer
balanceOf
owner
manualSend
manualSwap
payable
swapTokensForEth
approve
resAllFee
transferStandard
takeCom
tokenFromRef
getTValues
getRValues
getValues
getRate
getCurrentSupply
tAmount
sub
mul
tTeam
mul
rSupply
div
rTotal
div
tCom
mul
rAmount
sub
div
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
transferOwnership
_msgSender
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH

Summary

ØxLiquidity contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. ØxLiquidity is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract’s ownership has been renounced.

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