Cyberscope

Audit Report

Zap

March 2023

Network BSC

Address 0x2De0b483E9eaC4189F02E4c4d805302C3b52eb7a

Audited by © cyberscope

Table of Contents

Review

Explorer	https://bscscan.com/address/0x2de0b483e9eac4189f02e4c4d805302c3b52eb7a

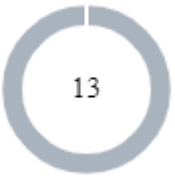
Audit Updates

Initial Audit	03 Apr 2023

Source Files

Filename	SHA256
ZapV2.sol	0d020935b86df0b33f7c12404e484ebccdf5cfb3c766fca58a31812609bf3175

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Introduction

The Zap2 smart contract implements a mechanism to simplify the conversion of one cryptocurrency into another. By sending one cryptocurrency, users can automatically receive an equivalent value of another. Zap contracts help to streamline the process of converting between cryptocurrencies.

Roles

The contract consists of the owner role.

The `owner` has the authority to:

- Configure contract parameters like native routers, bridge tokens, and contract fees.
- Withdraw for forgotten tokens and eth from the contract.

The `users` have the authority to:

- zapInToken : Adds liquidity to ETH by swapping half of the tokens for ETH, and then adding both the tokens and ETH to the liquidity pool.
- estimateZapInToken : getAmountOut through a path of tokens.
- zapIn : Adds eth to the liquidity.
- estimateZapIn: getAmount out for the tokens from the native tokens.
- zapAcross : Burns the transferred token from the contract and adds the burned token to liquidity.
- zapOut : removes liquidity from the liquidity pair.
- zapOutToken : remove tokens from the liquidity pair.
- swapToken : swaps tokens for tokens.
- swapToNative : swaps native to ETH.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	AETA	Approve Excessive Token Amounts	Unresolved
	PTAI	Potential Transfer Amount Inconsistency	Unresolved
	CR	Code Repetition	Unresolved
	MSE	Missing Solidity Events	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L18	Multiple Pragma Directives	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

AETA - Approve Excessive Token Amounts

Criticality	Minor / Informative
Location	ZapV2.sol#L1157
Status	Unresolved

Description

The contract uses the `_approveTokenIfNeeded` function approves a maximum amount of tokens to be transferred on behalf of the owner. This can potentially lead to security vulnerabilities, such as an attacker bypassing the intended limit by spending the entire approved amount in a single transaction.

```
function _approveTokenIfNeeded(address token, address router)
private {
    if (IERC20(token).allowance(address(this), router) == 0) {
        IERC20(token).safeApprove(router, type(uint).max);
    }
}
```

Recommendation

It's recommended to use the approve function to approve only the required amount of tokens instead of the maximum amount. This approach will ensure that the user's tokens are safe and will prevent unauthorized access.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	ZapV2.sol#L968,1047,1065,1102,1128,1139
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
function zapInToken(...){...}
function zapAcross(...){...}
function zapOut(...){...}
function zapOutToken(...){...}
function swapToken(...){...}
function swapToNative(...){...}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

CR - Code Repetition

Criticality	Minor / Informative
Location	ZapV2.sol#L1003,1195
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function _swapHalfNativeAndProvide(address token, uint amount,
address routerAddress, address recipient) private returns
(uint, uint, uint) {
    ...
    if (useNativeRouter[routerAddress]) {
        IHyperswapRouter01 router =
IHyperswapRouter01(routerAddress);
        return router.addLiquidityETH{value :
amount.sub(swapValue)}(token, tokenAmount, 0, 0, recipient,
block.timestamp);
    }
    else {
        IUniswapV2Router01 router =
IUniswapV2Router01(routerAddress);
        return router.addLiquidityETH{value :
amount.sub(swapValue)}(token, tokenAmount, 0, 0, recipient,
block.timestamp);
    }
}

if (token0 == WNATIVE || token1 == WNATIVE) {
    address token = token0 == WNATIVE ? token1 : token0;
    uint tokenAmt = _estimateSwap(WNATIVE, zapAmt, token,
_router);
    if (token0 == WNATIVE) {
        return (zapAmt, tokenAmt);
    } else {
        return (tokenAmt, zapAmt);
    }
}

// go through native token for highest liquidity
uint nativeAmount = _from == WNATIVE ? _amt :
_estimateSwap(_from, _amt, WNATIVE, _router);
return estimateZapIn(_to, _router, nativeAmount);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	ZapV2.sol
Status	Unresolved

Description

The contract has some actions that a user can take on the contract that do not emit any 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.

```
function zapInToken(...){...}
function zapAcross(...){...}
function zapOut(...){...}
function zapOutToken(...){...}
function swapToken(...){...}
function swapToNative(...){...}
```

Recommendation

It is recommended adding events to the code that are emitted whenever a user interacts with the contract in a significant way. 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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	ZapV2.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ZapV2.sol#L958
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`.

```
WNATIVE
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ZapV2.sol#L13,122,123,140,162,947,948,949,950,968,982,1008,1020,1047,1065,1102,1128,1139
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);
address private WNATIVE
address private FEE_TO_ADDR
uint16 FEE_RATE
uint16 MIN_AMT
address _from
address _recipient
address _to
uint _amt
address _router
address _LP

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	ZapV2.sol#L389,415,440,465,475,489,499,748,754,760,812,817
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls,
avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value,
recipient may have reverted");
...
function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level
call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ZapV2.sol#L958,1423
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.

```
WNATIVE = _WNATIVE
FEE_TO_ADDR = addr
```

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	ZapV2.sol#L369,516
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(account) }

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

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	ZapV2.sol#L7,847,873
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;
pragma solidity ^0.8.0;
```

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	ZapV2.sol#L847,873
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	ZapV2.sol#L1414
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(token).transfer(owner(),
IERC20(token).balanceOf(address(this)))
```

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**
IHyperswapRouter01	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		-
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		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IVault	Interface	IERC20		
	deposit	External	✓	-
	withdraw	External	✓	-
	want	External		-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	

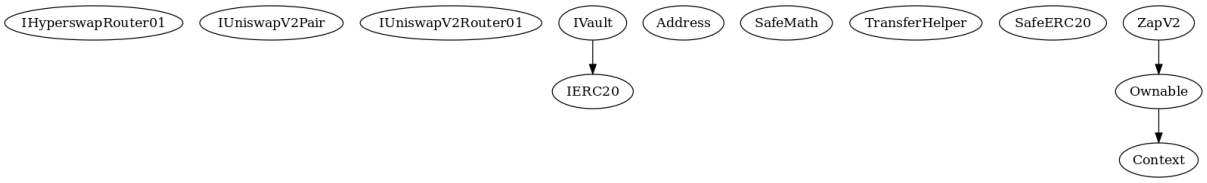
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		

	mod	Internal		
TransferHelper	Library			
	safeApprove	Internal	✓	
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeTransferETH	Internal	✓	
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-

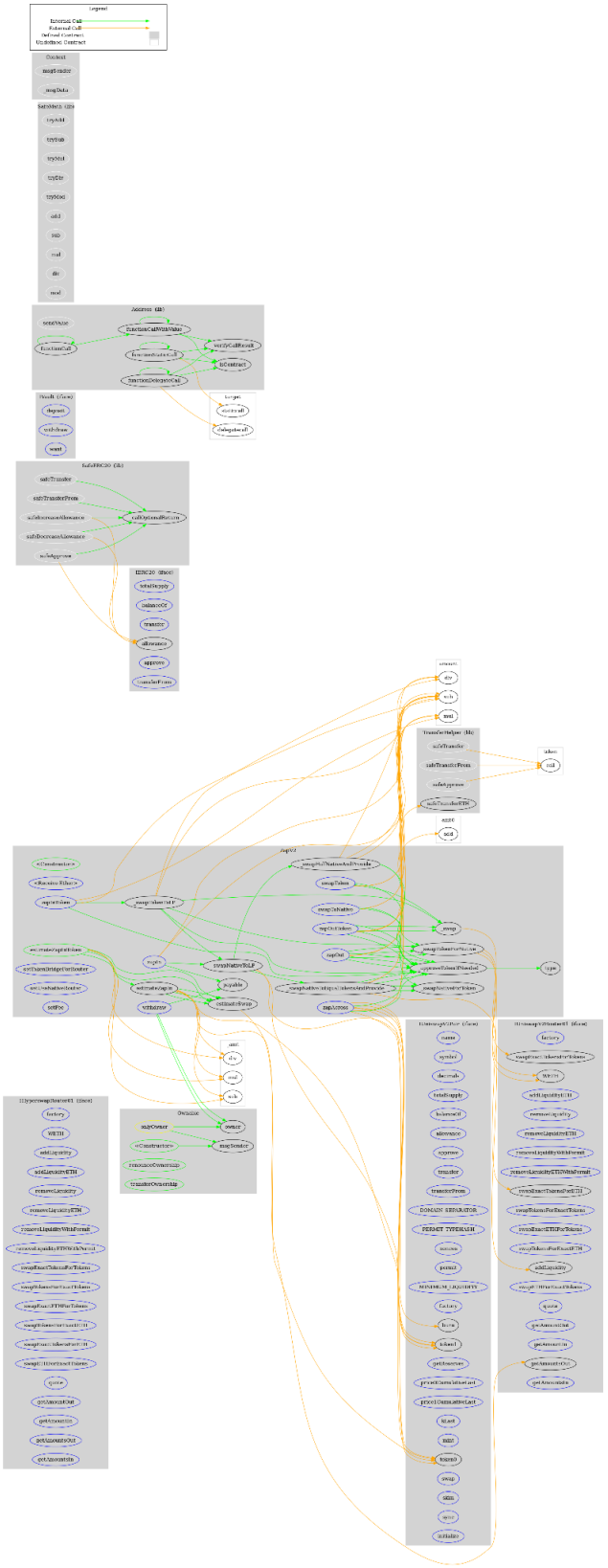
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
ZapV2	Implementation	Ownable		
		Public	✓	Ownable
		External	Payable	-
	zapInToken	External	✓	-
	estimateZapInToken	Public		-
	zapIn	External	Payable	-
	estimateZapIn	Public		-
	zapAcross	External	✓	-
	zapOut	External	✓	-
	zapOutToken	External	✓	-
	swapToken	External	✓	-
	swapToNative	External	✓	-
	_approveTokenIfNeeded	Private	✓	
	_swapTokenToLP	Private	✓	
	_swapNativeToLP	Private	✓	
	_swapHalfNativeAndProvide	Private	✓	
	_swapNativeToEqualTokensAndProvide	Private	✓	
	_swapNativeForToken	Private	✓	
	_swapTokenForNative	Private	✓	
	_swap	Private	✓	

	_estimateSwap	Private		
	setTokenBridgeForRouter	External	✓	onlyOwner
	withdraw	External	✓	onlyOwner
	setUseNativeRouter	External	✓	onlyOwner
	setFee	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

MyBricks contract implements a utility and financial 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