Cyberscope

Audit Report

The Tribe

November 2023

Network	ETH
Address	0xdeADFACE8503399Df4b083ef42FA8e02fd39dEAd
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
●	TSLS	Token Swap Logic Susceptibility	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	MU	Modifiers Usage	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	TheTribe
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	20000 runs
Explorer	https://etherscan.io/address/0xdeadface8503399df4b083ef42fa8e02fd39dead
Address	0xdeadface8503399df4b083ef42fa8e02fd39dead
Network	ETH
Symbol	TRIBE
Decimals	18
Total Supply	1,000,000,000

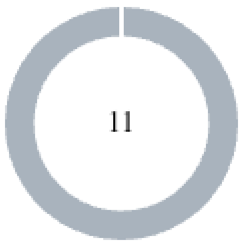
Audit Updates

Initial Audit	16 Nov 2023

Source Files

Filename	SHA256
contracts/TheTribe.sol	c269d1e069d98b97b1ee4911e121c1e1be1db615dcda83b4543d8c732742411d
contracts/ERC20UniswapV2InternalSwaps.sol	aa0635b1800613cb693bd15196e53739c05271d3fd58c4152022bdfa1fb501a7
@openzeppelin/contracts/utils/Context.sol	9c1cc43aa4a2bde5c7dea0d4830cd42c54813ff883e55c8d8f12e6189bf7f10a
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/ERC20.sol	2d874da1c1478ed22a2d30dcf1a6ec0d09a13f897ca680d55fb49fbcc0e0c5b1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	11

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	11	0	0	0

TSLS - Token Swap Logic Susceptibility

Criticality	Minor / Informative
Location	contracts/ERC20UniswapV2InternalSwaps.sol#L66,79,94,109,130,138,149
Status	Unresolved

Description

The contract presents a potential risk due to its reliance on specific assumptions about the behavior and internal mechanisms of the Uniswap router. The custom logic, encompassed within the contract's functions is built upon expectations regarding Uniswap router parameters, fees, and liquidity provision. This approach may introduce inconsistencies and could lead to transaction reverts in the future if the Uniswap router contract undergoes updates. The contract's functionality is tightly coupled with assumptions about the Uniswap router's current state, making it susceptible to changes in the underlying infrastructure.

```
/**
 * @dev Swap tokens to WETH directly on pair, to save gas.
 * No check for minimal return, susceptible to price manipulation!
 */
function _swapForWETH(uint amountToken, address to) internal {... }

/**
 * @dev Add tokens and WETH to liquidity, directly on pair, to save gas.
 * No check for minimal return, susceptible to price manipulation!
 * Sufficient WETH in contract balance assumed!
 */
function _addLiquidity(
    uint amountToken,
    address to
) internal returns (uint amountWeth) { ... }

/** @dev Quote `amountToken` in ETH, assuming no fees (used for
liquidity). */
function _quoteToken(
    uint amountToken
) internal view returns (uint amountEth) { ... }

/** @dev Quote `amountToken` in WETH, assuming 0.3% uniswap fees (used for
swap). */
function _getAmountWeth(
    uint amounToken
) internal view returns (uint amountWeth) { ... }
```

Recommendation

To ensure compatibility with future updates to the Uniswap router and avoid potential reverts, the team is advised to use the standard Uniswap router contract for token swapping. This approach ensures that the contract stays in sync with any changes made to the router's functionality, fees, and internal mechanisms. By relying on the standard and well-maintained Uniswap router, the contract can benefit from ongoing improvements and adjustments made by the Uniswap protocol developers, reducing the risk of unexpected behavior due to future updates.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/TheTribe.sol#L328
Status	Unresolved

Description

The contract tax manager has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `withdrawTaxes` function. If the function is called before the presale is over, then the participants will not be able to claim their tokens.

```
function withdrawTaxes() external onlyTaxManager {
    uint256 balance = balanceOf(address(this));
    if (balance > 0) {
        super._transfer(address(this), taxRecipient, balance);
        emit TaxesWithdrawn(balance);
    }
}
```

Recommendation

The team should carefully manage the private keys of the tax manager’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	contracts/TheTribe.sol#L287
Status	Unresolved

Description

The `setExchangePool` function allows the contract tax manager to set any arbitrary value without validation to the `isExchangePool` mapping, which is supposed to hold Uniswap pair addresses. This lack of validation can lead to unintended behavior, including the potential disruption of the contract's intended functionality.

```
function setExchangePool(
    address account,
    bool exchangePool
) external onlyTaxManager {
    if (isExchangePool[account] == exchangePool) {
        revert InvalidParameters();
    }
    isExchangePool[account] = exchangePool;
    emit ExchangePoolStateChanged(account, exchangePool);
}
```

Recommendation

The team should carefully manage the private keys of the tax manager’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	contracts/TheTribe.sol#L294,320
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

For instance, the `taxRecipient` address, as its name suggests, is the recipient of the fees collected by the contract. If this address is set to the zero address or an invalid address, the fees collected by the contract will be sent to an unrecoverable location, resulting in fund loss.

```
isExchangePool[account] = exchangePool;
taxRecipient = newTaxRecipient;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/TheTribe.sol#L146,172
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
if (_isContract(account)) {
    revert NoContract();
}
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	contracts/TheTribe.sol#L22
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
/** @notice Percentage of supply allocated for team, marketing, cex
listings, etc. (11.5%). */
uint256 public constant SHARE_OTHER = 11_50;
```

Recommendation

The team is advised to take these segments into consideration 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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/ERC20UniswapV2InternalSwaps.sol#L79,119,130
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 _addLiquidity(
        uint amountToken,
        address to
    ) internal returns (uint amountWeth) {
        amountWeth = _quoteToken(amountToken);
        _transferFromContractBalance(pair, amountToken);
        IERC20(WETH).transferFrom(address(this), pair, amountWeth);
        IUniswapV2Pair(pair).mint(to);
    }

function _sweepWeth(address to) internal returns (uint amountWeth) {
        amountWeth = IERC20(WETH).balanceOf(address(this));
        IERC20(WETH).transferFrom(address(this), to, amountWeth);
    }

...
```

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	contracts/TheTribe.sol#L111,113
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.

```
taxManager = _taxManager
taxRecipient = _taxRecipient
```

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	contracts/ERC20UniswapV2InternalSwaps.sol#L174,186
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 { // solhint-disable-line no-inline-assembly
            // Transfer the ETH and store if it succeeded or not.
            success := call(gas(), to, amount, 0, 0, 0, 0)
        }

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

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/TheTribe.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 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 ^0.8.20;
```

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/ERC20UniswapV2InternalSwaps.sol#L71,85,100,121
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(WETH).transferFrom(wethReceiver, to, amountWeth)
IERC20(WETH).transferFrom(address(this), pair, amountWeth)
IERC20(WETH).transferFrom(address(this), to, amountWeth)
```

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.

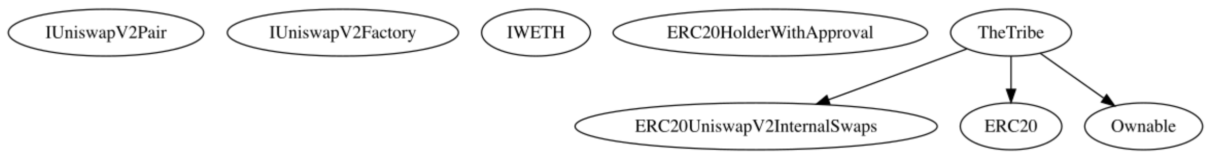
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TheTribe	Implementation	ERC20, Ownable, ERC20UniswapV2InternalSwaps		
		Public	✓	ERC20 Ownable
		External	Payable	-
	commitToPresale	Public	Payable	-
	claimPresale	External	✓	-
	unclaimedSupply	External		-
	openPresale	External	✓	onlyOwner
	closePresale	External	✓	onlyOwner
	completePresale	External	✓	onlyOwner
	setTaxFreeAccount	External	✓	onlyTaxManager
	setExchangePool	External	✓	onlyTaxManager
	transferTaxManager	External	✓	onlyTaxManager
	setTaxRecipient	External	✓	onlyTaxManager
	withdrawTaxes	External	✓	onlyTaxManager
	setSwapThresholdEth	External	✓	onlyTaxManager

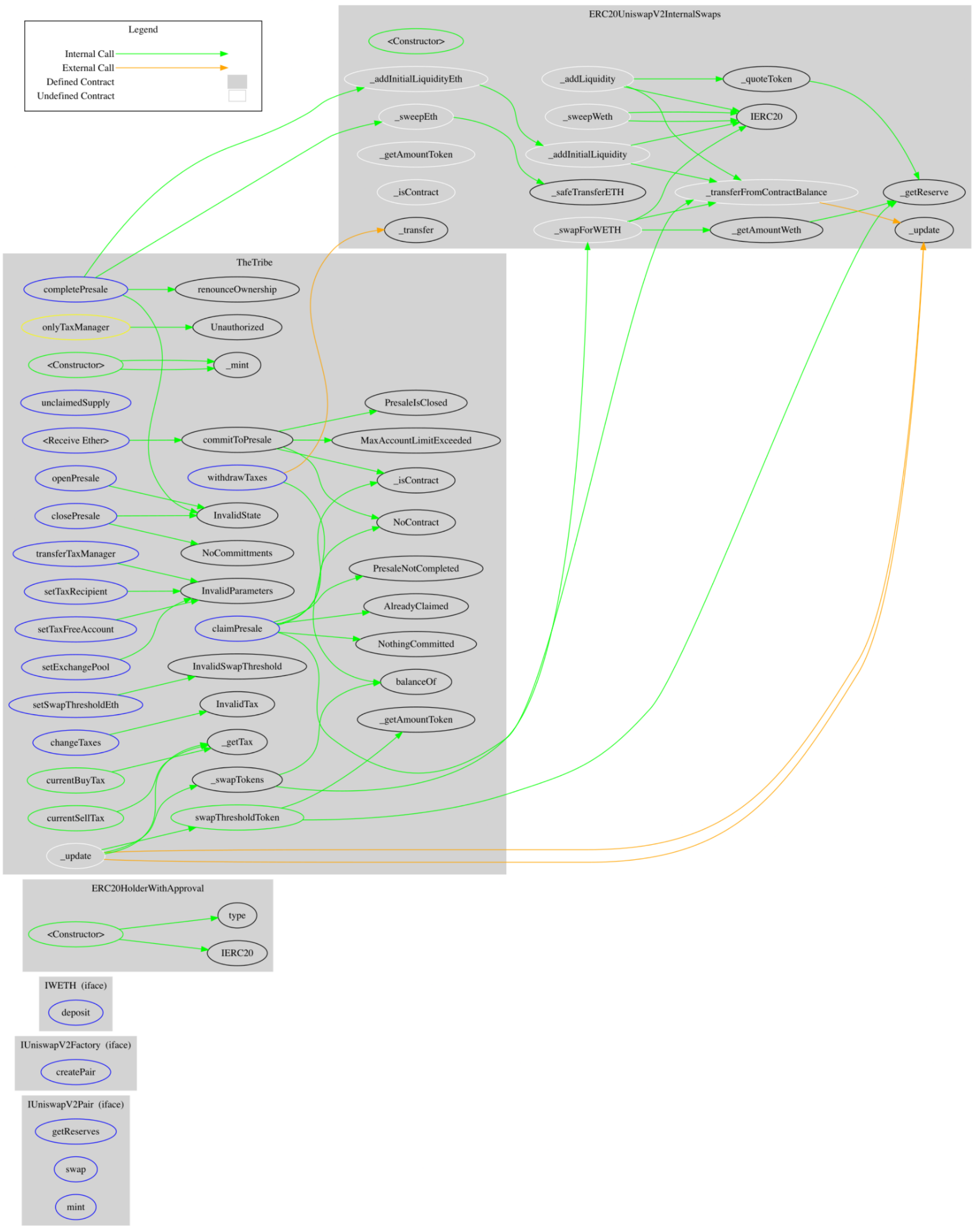
	changeTaxes	External	✓	onlyTaxManager
	swapThresholdToken	Public		-
	currentBuyTax	Public		-
	currentSellTax	Public		-
	_update	Internal	✓	
	_getTax	Private		
	_transferFromContractBalance	Internal	✓	
	_swapTokens	Internal	✓	
IUniswapV2Pair	Interface			
	getReserves	External		-
	swap	External	✓	-
	mint	External	✓	-
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IWETH	Interface			
	deposit	External	Payable	-
ERC20HolderWithApproval	Implementation			
		Public	✓	-

ERC20UniswapV2InternalSwaps	Implementation			
		Public	✓	-
	_swapForWETH	Internal	✓	
	_addLiquidity	Internal	✓	
	_addInitialLiquidity	Internal	✓	
	_addInitialLiquidityEth	Internal	✓	
	_sweepWeth	Internal	✓	
	_sweepEth	Internal	✓	
	_quoteToken	Internal		
	_getAmountWeth	Internal		
	_getAmountToken	Internal		
	_getReserve	Internal		
	_safeTransferETH	Internal	✓	
	_isContract	Internal		
	_transferFromContractBalance	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20UniswapV2InternalSwaps
<Constructor>
_addInitialLiquidityEth
_addLiquidity
_quoteToken
_sweepEth
_sweepWeth
IERC20
_getAmountToken
_addInitialLiquidity
_isContract
_safeTransferETH
_transferFromContractBalance
_getReserve
_transfer
_swapForWETH
_getAmountWeth
_update
TheTribe
completePresale
renounceOwnership
onlyTaxManager
Unauthorized
<Constructor>
_mint
unclaimedSupply
PresaleIsClosed
<Receive Ether>
commitToPresale
MaxAccountLimitExceeded
openPresale
withdrawTaxes
_isContract
closePresale
InvalidState
NoContract
transferTaxManager
NoCommittments
PresaleNotCompleted
setTaxRecipient
InvalidParameters
AlreadyClaimed
setTaxFreeAccount
claimPresale
NothingCommitted
setExchangePool
InvalidSwapThreshold
balanceOf
setSwapThresholdEth
InvalidTax
_getAmountToken
changeTaxes
_getTax
currentBuyTax
_swapTokens
currentSellTax
swapThresholdToken
_update
ERC20HolderWithApproval
type
<Constructor>
IERC20
IWETH (iface)
deposit
IUniswapV2Factory (iface)
createPair
IUniswapV2Pair (iface)
getReserves
swap
mint

Summary

The Tribe contract implements a token and presale mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The Tribe 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. There is also a limit of max 0.25% fees. Additionally, the contract incorporates a time-sensitive mechanism wherein a 3% fee is imposed on buys and a 50% fee on transfers and sales during the initial 20 minutes after launch.

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