Cyberscope

Audit Report

Magic Internet Toucans

February 2024

Network	ETH
Address	0x851De45ef74cBa4bC4f28E9038a49F71Be2A33B2
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	ZD	Zero Division	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RTC	Redundant Type Casting	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	MagicInternetToucans
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://etherscan.io/address/0x851de45ef74cba4bc4f28e9038a49f71be2a33b2
Address	0x851de45ef74cba4bc4f28e9038a49f71be2a33b2
Network	ETH
Symbol	MIT
Decimals	9
Total Supply	1,000,000,000
Badge Eligibility	Must Fix Criticals

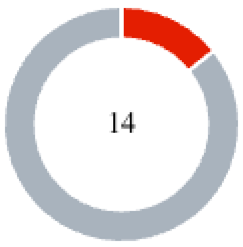
Audit Updates

Initial Audit	23 Feb 2024

Source Files

Filename	SHA256
MagicInternetToucans.sol	736c053a0097d3c9347436eba1bcd24aa3ce0d22cd0b2614a60210c65569c76c

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	12

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	MagicInternetToucans.sol#L265
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enabled the owner will not be able to disable them again.

```
if(!isFeeExempt[sender] &&
!isFeeExempt[recipient]){require(tradingAllowed, "tradingAllowed");}
```

Additionally, the contract owner has the authority to stop the sales for all users excluding the owner, as described in detail in sections ZD and PLPI. As a result, the contract might operate as a honeypot.

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

ZD - Zero Division

Criticality	Critical
Location	MagicInternetToucans.sol#L327
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 denominator = (liquidityFee.add(marketingFee)) * 2;
uint256 tokensToAddLiquidityWith =

tokens.mul(liquidityFee).div(denominator);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L208
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.

```
pair
```

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	MagicInternetToucans.sol#L269
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((_balances[recipient].add(amount)) <= _totalSupply)
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L279,291
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.

```
tradingAllowed = true;
isFeeExempt[user] = exempt;
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L325
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapAndLiquify(uint256 tokens) private lockTheSwap {
    uint256 denominator = (liquidityFee.add(marketingFee)) * 2;
    uint256 tokensToAddLiquidityWith =
tokens.mul(liquidityFee).div(denominator);
    uint256 toSwap = tokens.sub(tokensToAddLiquidityWith);
    uint256 initialBalance = address(this).balance;
    swapTokensForETH(toSwap);
    uint256 deltaBalance = address(this).balance.sub(initialBalance);
    uint256 unitBalance= deltaBalance.div(denominator.sub(liquidityFee));
    uint256 ETHToAddLiquidityWith = unitBalance.mul(liquidityFee);
    if(ETHToAddLiquidityWith > uint256(0)){
        addLiquidity(tokensToAddLiquidityWith, ETHToAddLiquidityWith); }
    uint256 remainingBalance = address(this).balance;
    if(remainingBalance > uint256(0)){
      payable(marketing_receiver).transfer(remainingBalance); }
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MagicInternetToucans.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 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 in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L279,291
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
tradingAllowed = true;
isFeeExempt[user] = exempt;
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RTC - Redundant Type Casting

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L260,275,318,334,337
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.

Specifically, the contract contains several instances where hardcoded integer values are explicitly cast to uint256, such as in conditions, require statements, and arithmetic operations. This practice of explicitly casting values, which are already implicitly cast to uint256, is redundant and could lead to unnecessary gas consumption during execution. Solidity automatically converts numeric literals without any suffix to the appropriate type, making such explicit casts unnecessary.

```
amount > uint256(0)
swapTimes += uint256(1)
swapTimes >= uint256(2)
ETHToAddLiquidityWith > uint256(0)
remainingBalance > uint256(0)
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L178,193
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 _totalSupply = 1000000000 * (10 ** _decimals)
bool swapEnabled = 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	MagicInternetToucans.sol#L137,175,176,177,179,180,200,201,341
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 = 'Magic Internet Toucans'
string private constant _symbol = 'MIT'
uint8 private constant _decimals = 9
uint256 public _maxTxAmount = ( _totalSupply * 10000 ) / 10000
uint256 public _maxWalletToken = ( _totalSupply * 10000 ) / 10000
address internal marketing_receiver = msg.sender
address internal liquidity_receiver = msg.sender
uint256 ETHAmount
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L299,332,333
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 feeAmount = amount.div(feeDenominator).mul(getTotalFee(sender,
recipient))
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L126
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 = adr
```

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	MagicInternetToucans.sol#L227
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(addr) }
```

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.

Functions Analysis

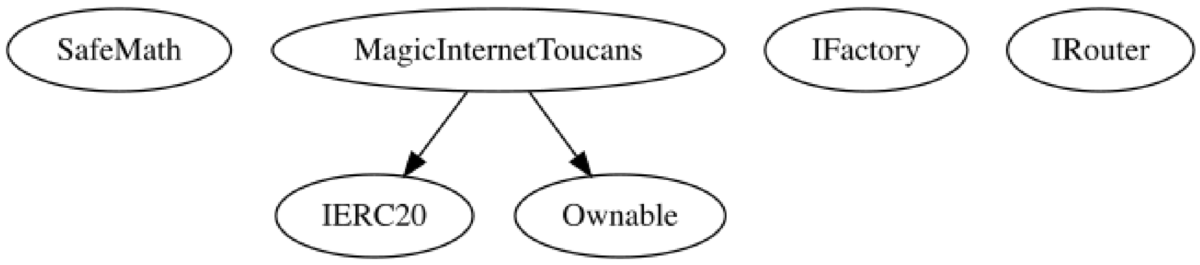
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-

	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation			
		Public	✓	-
	isOwner	Public		-
	transferOwnership	Public	✓	onlyOwner
IFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	removeLiquidityWithPermit	External	✓	-

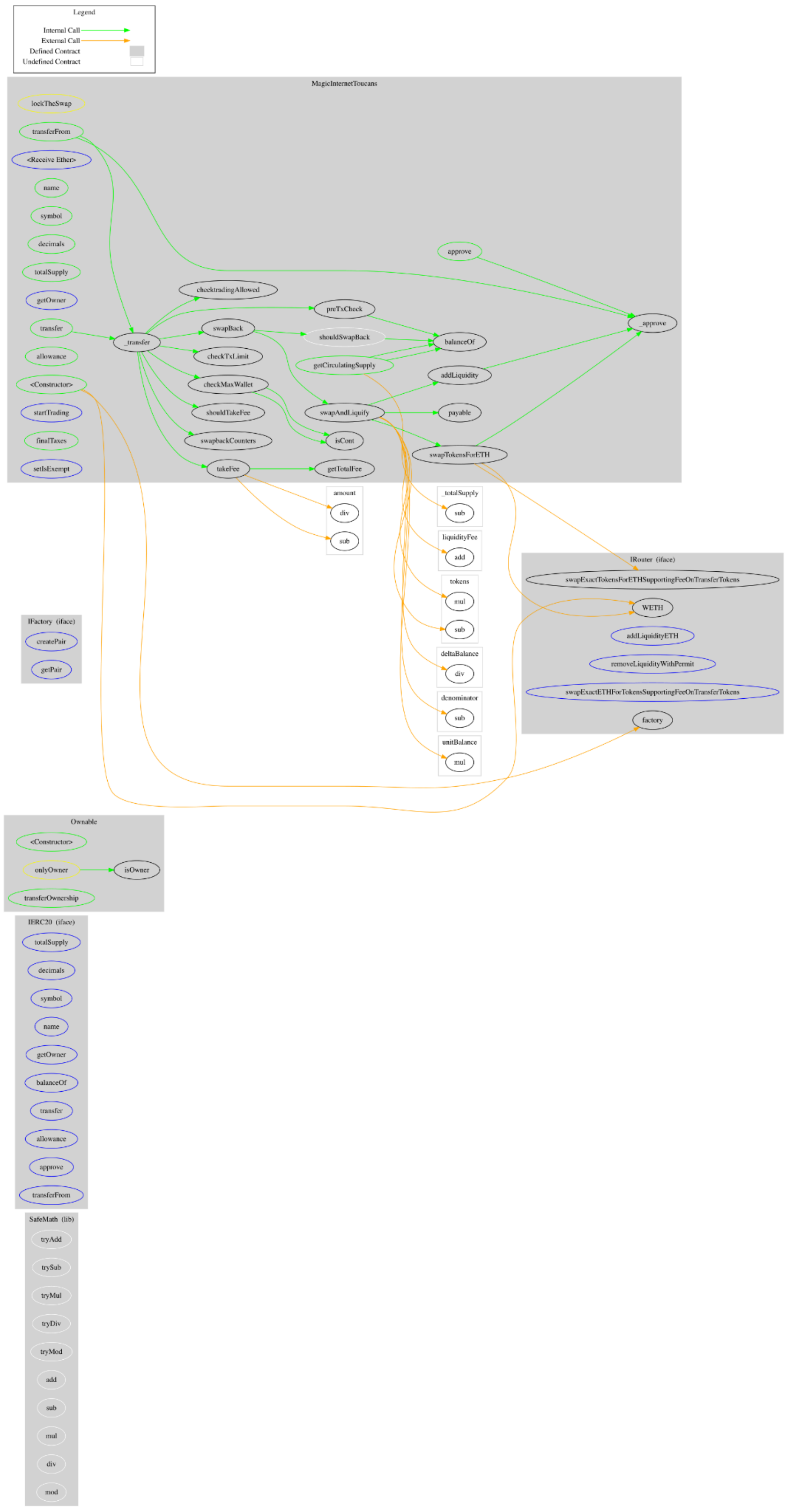
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
MagicInternetToucans	Implementation	IERC20, Ownable		
		Public	✓	Ownable
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	getOwner	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	isCont	Internal		
	approve	Public	✓	-
	getCirculatingSupply	Public		-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	preTxCheck	Internal		
	checktradingAllowed	Internal		
	checkMaxWallet	Internal		

	swapbackCounters	Internal	✓	
	startTrading	External	✓	onlyOwner
	finalTaxes	Public	✓	onlyOwner
	setIsExempt	External	✓	onlyOwner
	shouldTakeFee	Internal		
	takeFee	Internal	✓	
	getTotalFee	Internal		
	checkTxLimit	Internal		
	shouldSwapBack	Internal		
	swapBack	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	addLiquidity	Private	✓	
	swapTokensForETH	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
MagicInternetToucans
lockTheSwap
transferFrom
<Receive Ether>
name
symbol
decimals
totalSupply
getOwner
transfer
allowance
<Constructor>
startTrading
finalTaxes
setIsExempt
_transfer
checktradingAllowed
preTxCheck
swapBack
shouldSwapBack
checkTxLimit
getCirculatingSupply
checkMaxWallet
shouldTakeFee
swapbackCounters
takeFee
getTotalFee
swapAndLiquify
isCont
approve
_approve
balanceOf
addLiquidity
payable
swapTokensForETH
amount
div
sub
_totalSupply
liquidityFee
add
tokens
mul
deltaBalance
denominator
unitBalance
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
removeLiquidityWithPermit
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
IFactory (iface)
createPair
getPair
Ownable
onlyOwner
isOwner
transferOwnership
IERC20 (iface)
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
mod

Summary

Magic Internet Toucans contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 20% fees.

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