Cyberscope

Audit Report

Magic Internet Toucans

February 2024

Network	ETH
Address	0xbfab8e3310677d9719591e5f439456799563eee0
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
●	IDI	Immutable Declaration Improvement	Unresolved
●	IFS	Inefficient Fee Structure	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RFV	Redundant Fee Variable	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RTL	Redundant Transaction Limits	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	MagicInternetToucans
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://etherscan.io/address/0xbfab8e3310677d9719591e5f439456799563eee0
Address	0xbfab8e3310677d9719591e5f439456799563eee0
Network	ETH
Symbol	MIT
Decimals	9
Total Supply	1,000,000,000
Badge Eligibility	Yes

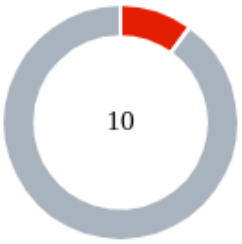
Audit Updates

Initial Audit	23 Feb 2024 https://github.com/cyberscope-io/audits/blob/main/4-mit/v1/audit.pdf
Corrected Phase 2	25 Feb 2024

Source Files

Filename	SHA256
MagicInternetToucans.sol	8e552ad7bf02a7fd78cb4b35a58bf343c12b7b7cfbe78eaca4dad5ed5b57edf1

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	MagicInternetToucans.sol#L266
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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L207
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.

```
marketing_receiver
```

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.

IFS - Inefficient Fee Structure

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L191
Status	Unresolved

Description

The contract is utilizing four different fee variables, specifically the `marketingFee`, `totalFee`, `sellFee`, and `transferFee`, each set to a value of `2000`. This setup implies an intention to differentiate between various types of fees for distinct operations within the contract. However, the uniformity in the value of these fees, all being equal to 2000, in combination with the absence of any functionality to update these fee variables, leads to an inefficient and redundant fee structure. Without the ability to modify these fees, the contract lacks the flexibility to adapt its fee logic to different circumstances or objectives. This rigidity effectively renders the distinction between the four fee types meaningless, as they all impose the same charge. Consequently, the current fee logic does not leverage the potential benefits of having multiple fee variables, such as the ability to apply differentiated fees on the overall transaction process.

```
uint256 marketingFee = 2000;
uint256 totalFee = 2000;
uint256 sellFee = 2000;
uint256 transferFee = 2000;
```

Recommendation

It is recommended to reconsider the fee logic of the contract. Given that all fees are set to the same value and cannot be updated, the contract could be refactored to utilize only one fee variable. This simplification would not only reduce the contract's complexity but also enhance its efficiency by eliminating unnecessary variables. If differentiated fee structures are desired in the future, it is advisable to implement a mechanism for dynamically updating fee values. This approach would provide the necessary flexibility to adjust fees in response to evolving requirements or strategies, thereby making the fee logic more effective and adaptable.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L337
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 swapTokensForETH(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        _approve(address(this), address(router), tokenAmount);
        router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp);
    }
```

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.

RFV - Redundant Fee Variable

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L191
Status	Unresolved

Description

The contract is currently declaring the `marketingFee` variable with a value of `2000`. However the `marketingFee` variable is not integrated into any of the contract's operational or fee calculation processes. This oversight leads to the variable being redundant within the contract's architecture. The presence of such unused variables not only consumes unnecessary space but also introduces potential confusion regarding the contract's intended functionalities and fee structure. The absence of `marketingFee` in the contract's fee logic indicates a misalignment between the declared variables and their practical application within the contract.

```
uint256 marketingFee = 2000;
```

Recommendation

It is recommended to remove the redundant `marketingFee` variable from the contract. This action would streamline the contract's codebase, eliminating unnecessary elements that do not contribute to the contract's functionality. Removing the variable would also enhance the clarity and maintainability of the contract. Furthermore, this step would ensure that the contract's resources are optimized, focusing solely on variables and functions that serve a defined purpose within the contract's operations.

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.

```
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.

RTL - Redundant Transaction Limits

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L183,269,316
Status	Unresolved

Description

The contract is implementing the `checkTxLimit` and `checkMaxWallet` functions to enforce limits on transaction amounts and wallet token holdings, respectively. These functions rely on `_maxTxAmount` and `_maxWalletToken` variables, intended to set caps on the maximum transaction size and the maximum number of tokens a wallet can hold. However, both variables are set to equal the contract's total supply, effectively rendering these checks redudant. As a result, the intended restrictions do not apply, since the limits are set to the maximum of the `_totalSupply`. This configuration negates the purpose of having transfer and wallet holding limits, as all transactions and wallet balances below or equal to the total supply are permitted without any actual restriction.

```
   uint256 public constant _maxTxAmount = ( _totalSupply * 10000 ) /
10000;
    uint256 public constant _maxWalletToken = ( _totalSupply * 10000 ) /
10000;

    function checkTxLimit(address sender, address recipient, uint256
amount) internal view {
        require(amount <= _maxTxAmount || isFeeExempt[sender] ||
isFeeExempt[recipient], "TX Limit Exceeded");
    }

    function checkMaxWallet(address sender, address recipient, uint256
amount) internal view {
        if(!isFeeExempt[sender] && !isFeeExempt[recipient] && recipient
!= address(pair) && recipient != address(DEAD)){
            require((_balances[recipient].add(amount)) <=
_maxWalletToken, "Exceeds maximum wallet amount.");}
    }
```

Recommendation

It is recommended to reconsider the functionality of the `checkTxLimit` and `checkMaxWallet` functions. If the intended functionality is indeed to apply meaningful limits to transactions and wallet holdings, then `_maxTxAmount` and `_maxWalletToken` should be set to a percentage of `_totalSupply` that is lower than 100%. This adjustment would ensure that the functions serve their purpose by effectively imposing restrictions that enhance the contract's security and manageability. By recalibrating these limits to reflect a genuine fraction of the total supply, the contract can better control token distribution and transaction volumes, aligning with common practices for preventing manipulation and ensuring equitable token distribution among holders.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MagicInternetToucans.sol#L194,195
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 swapThreshold = ( _totalSupply * 500 ) / 100000
uint256 _minTokenAmount = ( _totalSupply * 50 ) / 100000
```

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#L137,175,176,177,178,181,195,197
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 private constant _totalSupply = 1000000000 * (10 ** _decimals)
mapping (address => uint256) _balances
uint256 _minTokenAmount = ( _totalSupply * 50 ) / 100000
address internal marketing_receiver
```

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.

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.

Functions Analysis

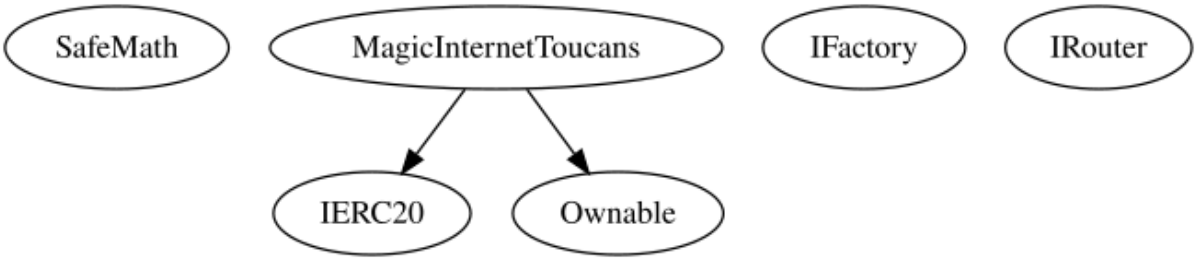
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal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

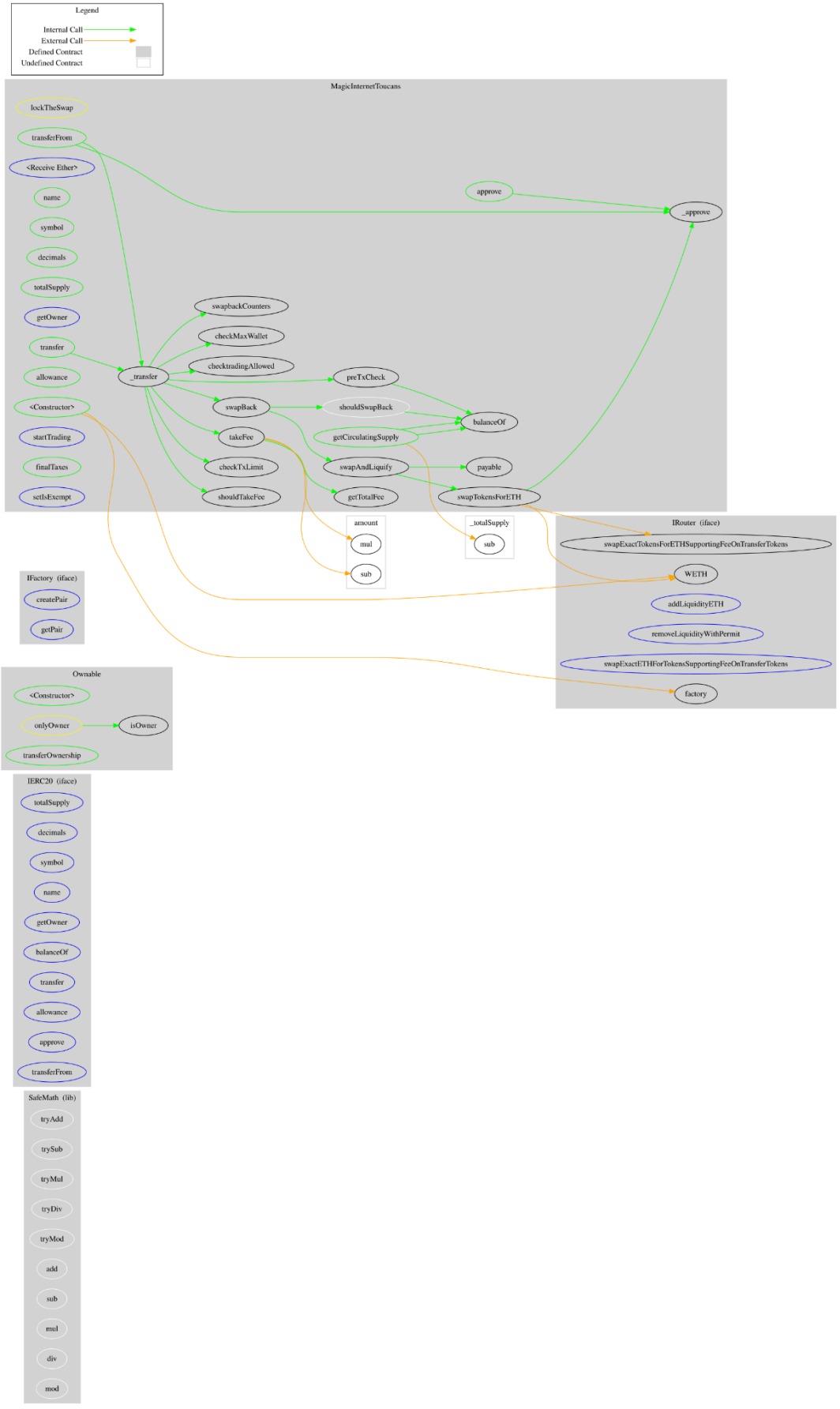
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
MagicInternetToucans	Implementation	IERC20, Ownable		
		Public	✓	Ownable
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	getOwner	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	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
	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
approve
_approve
swapbackCounters
checkMaxWallet
checktradingAllowed
_transfer
preTxCheck
swapBack
shouldSwapBack
takeFee
getCirculatingSupply
balanceOf
checkTxLimit
swapAndLiquify
payable
shouldTakeFee
getTotalFee
swapTokensForETH
amount
mul
sub
_totalSupply
sub
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
removeLiquidityWithPermit
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
IFactory (iface)
createPair
getPair
Ownable
<Constructor>
onlyOwner
isOwner
transferOwnership
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
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 25% 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