Cyberscope

Audit Report

GuiaMia

May 2024

Network	BSC
Address	0x948B6e0eBb4655973935aA4633447A70e5A6f473
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
●	DDP	Decimal Division Precision	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved

●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	GuiaMia
Testing Deploy	https://bscscan.com/address/0x948B6e0eBb4655973935aA4633447A70e5A6f473
Symbol	CLIP
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Must Fix Critical, Proxy Contract Ownership should be Renounced

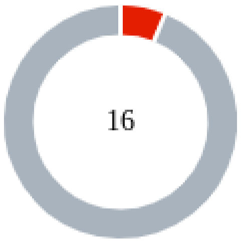
Audit Updates

Initial Audit	21 May 2024

Source Files

Filename	SHA256
Utils.sol	b9cafc74f68dd72a7c9fe44db77bac1d0f7d4fe7bd37e044243030e81bbfdad7
Uniswap.sol	2737aa98fb6dbfdf508b1d4c7cbc92413dc4cdd266ceb8a1b22ba9685e1b0fd8
TransparentUpgradable.sol	a3aeb03af1a7f9424dd366762dd0dea55b4d430b3772cd01dc0e27dda605241d
GuiaMia.sol	1a4189885fd84fd3f875f5dc6108c7c4cc318c1b9670311f593569ff936e6ce4

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	15

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	GuiaMia.sol
Status	Unresolved

Description

the contract owner has the authority to stop transactions, as described in detail in sections `PMRM`. As a result, the contract might operate as a honeypot.

Recommendation

The team is advised to follow the recommendations outlined in the `PRPM` finding and implement the necessary steps to mitigate the identified risk, ensuring that the contract does not operate as a honeypot.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	GuiaMia.sol#L381
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
safeTransferETH(
    developmentAddress,
    _calcuclateShare(tokenConfigs.coinShareDevelopment,
balance)
);
safeTransferETH(
    lpVaultAddress,
    _calcuclateShare(tokenConfigs.coinShareLP, balance)
);
safeTransferETH(
    marketingAddress,
    _calcuclateShare(tokenConfigs.coinShareMarketing, balance)
);
safeTransferETH(
    coinRewardsAddress,
    _calcuclateShare(tokenConfigs.coinShareRewards, balance)
);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	TransparentUpgradable.sol#L115,157
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(success)
```

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.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	GuiaMia.sol#L430
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.

```
_balances[address(this)] =
_balances[address(this)].add(taxAmount);
```

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	GuiaMia.sol#L338
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.

```
address pair = factory.getPair(path[0], path[1]);
uint256 maxSwap = _balances[pair].div(100);

swappedAmount = tokenAmount > maxSwap ? maxSwap : tokenAmount;
buyBackReserveShare = swappedAmount
    .mul(tokenConfigs.buyBackShareReserve)
    .div(100);

swappedAmount = swappedAmount.sub(buyBackReserveShare);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    swappedAmount,
    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.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	GuiaMia.sol#L648
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function setupExchange(address newRouter) external onlyOwner {
      _setupExchange(newRouter);
}
```

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.

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.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	GuiaMia.sol#L68
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event EnabledUniswap();
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Utils.sol GuiaMia.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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Uniswap.sol#L68,70,100,140 GuiaMia.sol#L466,475,487,502,510,525,589,595,600,605
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function WETH() external pure returns (address);
address _addr
address payable _marketingAddress
address payable _developmentAddress
address payable _vaultAddress
address payable _coinRewardsAddress
address _tokenReserveAddress
uint256 _minimumTokensBeforeSwap
uint256 _minimumETHToTransfer
bool _enabled
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	GuiaMia.sol#L591,597,612
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
minimumTokensBeforeSwap = _minimumTokensBeforeSwap
minimumETHToTransfer = _minimumETHToTransfer
autoSplitShares = _enabled
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Utils.sol#L109,119,132,139,147,161,174 GuiaMia.sol#L288,298
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 isContract(address account) internal view returns
(bool) {
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            codehash := extcodehash(account)
        }
        return (codehash != accountHash && codehash != 0x0);
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	GuiaMia.sol#L343,345,346
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 maxSwap = _balances[pair].div(100)
buyBackReserveShare = swappedAmount
            .mul(tokenConfigs.buyBackShareReserve)
            .div(100)
swappedAmount = tokenAmount > maxSwap ? maxSwap : tokenAmount
```

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	TransparentUpgradable.sol#L114,156
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.

```
(bool success, ) = logic.delegatecall(data)
(bool success, ) = newImplementation.delegatecall(data)
```

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	Utils.sol#L113,189 TransparentUpgradable.sol#L17,55,70,88,162,170 GuiaMia.sol#L468
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 {
            codehash := extcodehash(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	Utils.sol#L2 Uniswap.sol#L2 TransparentUpgradable.sol#L1 GuiaMia.sol#L16
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 ^0.8.16;
pragma solidity ^0.8.9;
```

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	Utils.sol#L2 Uniswap.sol#L2 TransparentUpgradable.sol#L1 GuiaMia.sol#L16
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 ^0.8.16;
pragma solidity ^0.8.9;
```

Recommendation

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

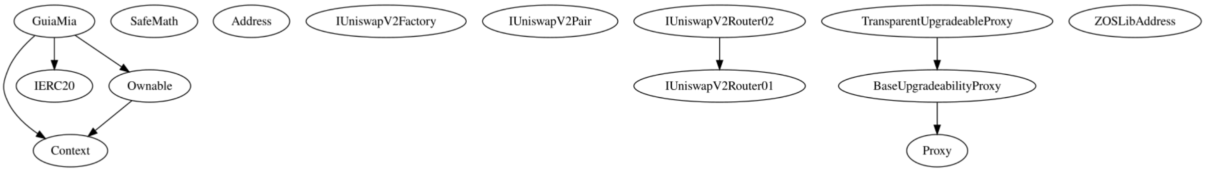
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
GuiaMia	Implementation	Context, IERC20, Ownable		
	initialize	Public	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	handleTaxAutomation	Internal	✓	
	safeTransferETH	Internal	✓	
	safeTransferToken	Internal	✓	
	safeTransferFrom	Internal	✓	

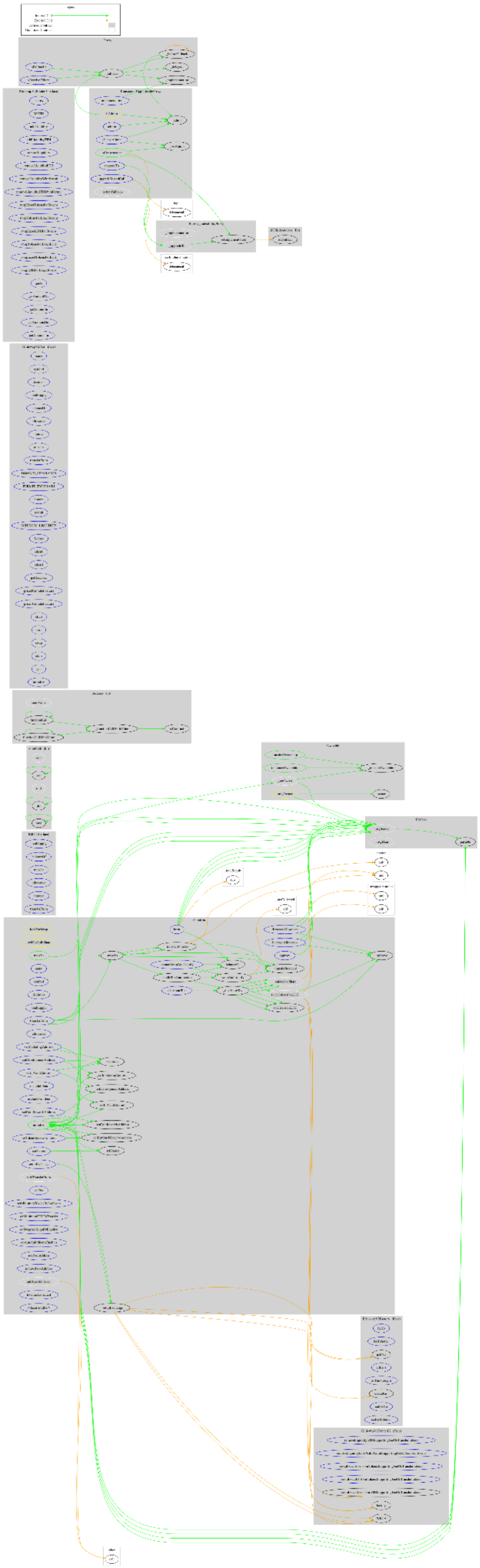
	manualSwapAndLiquify	External	✓	onlyOwner
	swapTokensForETH	Internal	✓	
	swapAndLiquify	Internal	✓	lockForSwap
	_calcuclateShare	Internal		
	_distributeTax	Internal	✓	lockForSplitShare
	distributeTax	External	✓	onlyOwner
	_transferStandard	Internal	✓	
	_transferWithTax	Internal	✓	
	includeInFee	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	_setMarketingAddress	Internal	✓	
	_setDevelopmentAddress	Internal	✓	
	_setLpVaultAddress	Internal	✓	
	_setCoinRewardsAddress	Internal	✓	
	_setBuyBackReserveAddress	Internal	✓	
	isContract	Internal		
	setMarketingAddress	External	✓	onlyOwner
	setDevelopmentAddress	External	✓	onlyOwner
	setLpVaultAddress	External	✓	onlyOwner
	setCoinRewardsAddress	External	✓	onlyOwner
	setTokenReserveAddress	External	✓	onlyOwner
	_setShares	Internal	✓	
	setShares	External	✓	onlyOwner
	getTax	External		-

	setMinimumTokensBeforeSwap	External	✓	onlyOwner
	setMinimumETHToTransfer	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	setAutoSplitSharesEnables	External	✓	onlyOwner
	addPoolAddress	External	✓	onlyOwner
	removePoolAddress	External	✓	onlyOwner
	_setupExchange	Internal	✓	
	setupExchange	External	✓	onlyOwner
	totalTaxCollected	External		onlyOwner
	burn	External	✓	-
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

GuiaMia 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% fee on buy and sell transactions.

Initial Audit, 21 May 2024

At the time of the audit report, the contract with address 0x948B6e0eBb4655973935aA4633447A70e5A6f473 is pointed out by the following proxy address: 0x5Ba95539F0657Fd79F0a5AffD855deAec621Df08.

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