Cyberscope

Audit Report

RWA IMMO

June 2024

Network	BSC
Address	0xE0844779f9BFe134A96FFcf9850E2A69F4Be5D57
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Unresolved
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Passed

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	CSD	Circulating Supply Discrepancy	Unresolved
🟠	TFD	Transfer Functions Distinction	Unresolved
⚪	RRS	Redundant Require Statement	Unresolved
⚪	RSML	Redundant SafeMath Library	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L05	Unused State Variable	Unresolved
⚪	L09	Dead Code Elimination	Unresolved
⚪	L16	Validate Variable Setters	Unresolved
⚪	L18	Multiple Pragma Directives	Unresolved
⚪	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	RWAIMMO
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://bscscan.com/address/0xe0844779f9bfe134a96ffcf9850e2a69f4be5d57
Address	0xe0844779f9bfe134a96ffcf9850e2a69f4be5d57
Network	BSC
Symbol	RWAIMMO
Decimals	18
Total Supply	5,000,000,000
Badge Eligibility	Must Fix Criticals

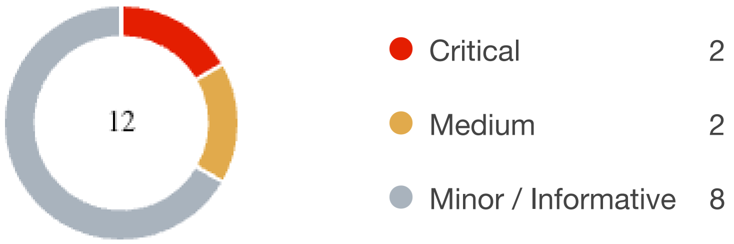
Audit Updates

Initial Audit	10 Jun 2024

Source Files

Filename	SHA256
RWAIMMO.sol	02ee5c7aefa0a0151688fa822c66ed7b7aae4e369d8cc59075b03fb2c7d479b5

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	2	0	0	0
● Medium	2	0	0	0
● Minor / Informative	8	0	0	0

OTUT - Transfers User's Tokens

Criticality	Critical
Location	RWAIMMO.sol#L1344,1370
Status	Unresolved

Description

The contract owner has the authority to transfer the balance of a user's address to the `STAKING_LOCKED_ADDRESS` 's address or the `AGENCY_LOCKED_ADDRESS` 's address. The owner may take advantage of it by calling either of the following functions:

- `transferStake`
- `transferBuy`

```
function transferStake(address _from, uint256 _value)
    public
    isContractOwner
    nonReentrant
    returns (bool)
{
    _stake(_from, STAKING_LOCKED_ADDRESS, _value);

    emit TransferStake(_from, STAKING_LOCKED_ADDRESS, _value);

    return true;
}
...
```

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.

BT - Burns Tokens

Criticality	Critical
Location	RWAIMMO.sol#L1427
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `payPenality` function. As a result, the targeted address will lose the corresponding tokens.

```
function payPenality(address _from, uint256 _value)
    public
    isContractOwner
    nonReentrant
    returns (bool)
{
    require(
        _value != 0 && _value <= _balances[_from],
        "ERR9"//"No coin in balance"
    );

    _balances[_from] = _balances[_from].sub(_value);
    _balances[BURN_ADDRESS] = _balances[BURN_ADDRESS].add(_value);

    _circulatingSupply = _circulatingSupply.sub(_value);

    emit PayPenality(_from, BURN_ADDRESS, _value);
    emit Transfer(_from, BURN_ADDRESS, _value);

    return true;
}
```

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.

CSD - Circulating Supply Discrepancy

Criticality	Medium
Location	RWAIMMO.sol#L1262
Status	Unresolved

Description

According to the ERC20 specification, the `totalSupply()` function should return the total supply of the token. The total supply should always equal the sum of the balances. The contract does not return the `totalSupply()` . Instead, the function returns the `totalSupply()` minus the amount that has been moved to the dead address. This amount is the circulating supply of the token. Many decentralized applications and tools are calculating many indicators like the circulating supply and market cap based on the `totalSupply()` . Additionally, the contract includes the `circulatingSupply` function, which should return the circulating supply if of the token. However, this function returns the value of the total supply minus the balance of the `STAKING_REWARDS_ADDRESS` . As a result, these applications will produce misleading results.

```
function totalSupply() public view override returns (uint256) {
    return _totalSupply - _balances[BURN_ADDRESS];
}
```

Recommendation

The `totalSupply()` should always equal the sum of the holder's balances. The contract should comply with this convention so that the decentralized applications will produce correct results.

TFD - Transfer Functions Distinction

Criticality	Medium
Location	RWAIMMO.sol#L1313
Status	Unresolved

Description

The `transfer` and `transferFrom` functions of an ERC20 token are used to transfer tokens from one user to another. The `transfer` function is augmented with additional features, such as restrictions on the sender wallet. Conversely, the `transferFrom` function adheres to the standard ERC20 transfer mechanism, focusing solely on transferring the specified amount from the sender to the recipient, with allowance checks to ensure the transaction does not exceed the sender's permitted amount.

This discrepancy might introduce complexity and potential confusion for users and external contracts interacting with the token, as the effects and outcomes of transferring tokens depend on the method used. For example, a `transfer` operation might result in a transaction revert if the sender is one of the blacklisted addresses, while a `transferFrom` operation would bypass these mechanisms entirely.

```
function transfer(address to, uint256 value)
    public
    override
    returns (bool)
{
    address owner = _msgSender();

    if (owner == BURN_ADDRESS || owner == STAKING_REWARDS_ADDRESS || owner
== STAKING_LOCKED_ADDRESS || owner == AGENCY_LOCKED_ADDRESS) {
        revert ERC20InvalidSender(owner);
    }

    _transfer(owner, to, value);
    return true;
}
```

Recommendation

To address this inconsistency and ensure a uniform experience for all token transfers, the team is advised to align the functionalities of `transfer` and `transferFrom` functions by either incorporating the additional mechanisms (fee, restrictions, etc.) into both functions or simplifying the transfer function to match the standard behavior observed in transferFrom. The choice should be based on the token's intended use case and economic model. By aligning the functionalities of both functions, the contract can provide a consistent and secure experience for token transfers, aligning with user expectations and the principles of the ERC20 standard, while still supporting the token's unique economic model and features.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	RWAIMMO.sol#L1191
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
  c = a + b;
  assert(c >= a);
  return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	RWAIMMO.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	RWAIMMO.sol#L393,788,792,797,798,832,843,844,845,858,867,868,874,890,911,919,926,930,1078,1086,1105,1126,1235,1332,1344,1357,1370,1383,1396,1427,1449
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.

```
mapping(address account => uint256) public _balances;
uint part_quantity
uint min_part
uint max_part
address _investor
uint _home_id
uint _quantity
...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	RWAIMMO.sol#L1235
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
mapping (address => uint256 ) _ETHBalances;
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	RWAIMMO.sol#L189,584,599
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 _contextSuffixLength() internal view virtual returns (uint256) {
        return 0;
    }

function _mint(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidReceiver(address(0));
        }
        _update(address(0), account, value);
    }
...
```

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	RWAIMMO.sol#L41,833,1079
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 = newOwner
_agencyAddress = _address;
_stakingAddress = _address;
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	RWAIMMO.sol#L4,58,140,168,198,363,679,712,809,938,988,1052,1150,1200,1505
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 >=0.7.0 <0.9.0;
pragma solidity ^0.8.20;
pragma solidity >0.7.0 <0.9.0;
pragma solidity ^0.8.0;
...
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	RWAIMMO.sol#L58,140,168,198,363,988
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;
pragma solidity ^0.8.0;
```

Recommendation

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

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Owner	Implementation			
		Public	✓	-
	changeOwner	Public	✓	isOwner
	getOwner	Public		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
ERC20	Implementation	Context, IERC20, IERC20Metadata, IERC20Errors		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-

	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
IRWAERC20	Interface			
	availableForStakingRewards	External		-
	burned	External		-
	totalStaked	External		-
	circulatingSupply	External		-
	transferStake	External	✓	-
	transferUnstake	External	✓	-
	transferBuy	External	✓	-
	transferSell	External	✓	-
	payClaim	External	✓	-
	payPenality	External	✓	-
IRWAAgency	Interface			
	totalInvesting	External		-
	investorsByPeriods	External		-

	investOf	External		-
	investOfByHome	External		-
	countInvestors	External		-
	addHome	External	✓	-
	getHomes	External		-
	getCountHomes	External		-
	getOwnerParts	External		-
	buy	External	✓	-
	sell	External	✓	-
	claim	External	✓	-
	rewards	External		-
	invested	External		-
	activate	External	✓	-
	isActive	External		-
RWAAgencyContractInit	Implementation	Owner		
	getAgencyContract	Public		-
	setAgencyContract	Public	✓	isOwner
	agencyAddHome	External	✓	isOwner
	agencyGetHomes	External		-
	agencyGetCountHomes	External		-
	agencyGetOwnerParts	External		-
	agencyBuy	External	✓	-

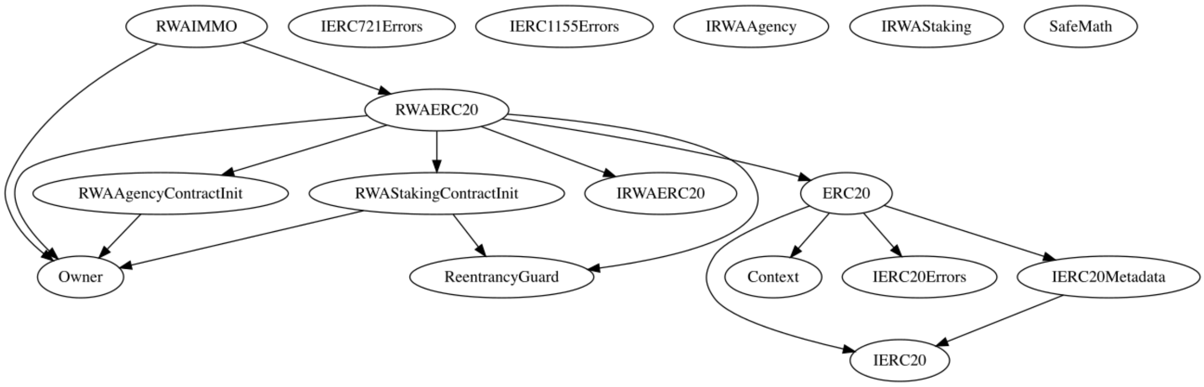
	agencySell	External	✓	-
	agencyRewards	Public		-
	agencyInvested	Public		-
	agencyClaim	Public	✓	-
	agencyTotalInvesting	Public		-
	agencyInvestorsByPeriods	Public		-
	agencyInvestOf	Public		-
	agencyCountInvestors	Public		-
	agencyActivate	External	✓	isOwner
	agencyIsActive	External		-
	agencyInvestOfByHome	Public		-
IRWAStaking	Interface			
	totalStaking	External		-
	stakesByPeriods	External		-
	stakeOf	External		-
	countStakers	External		-
	maxStakingPerAddress	External		-
	maxStakeTotal	External		-
	minStakingPerAddress	External		-
	stake	External	✓	-
	unstake	External	✓	-
	rewards	External		-

	staked	External		-
	claim	External	✓	-
ReentrancyGuard	Implementation			
		Public	✓	-
RWAStakingContractInit	Implementation	Owner, ReentrancyGuard		
	getStakingAddress	Public		-
	setStakingContract	Public	✓	isOwner
	stakingStake	Public	✓	-
	stakingUnstake	Public	✓	-
	stakingRewards	Public		-
	stakingStaked	Public		-
	stakingClaim	Public	✓	-
	stakingTotalStaking	Public		-
	stakingStakesByPeriods	Public		-
	stakingStakeOf	Public		-
	stakingCountStakers	Public		-
	stakingMaxStakingPerAddress	Public		-
	stakingMaxStakeTotal	Public		-
	stakingMinStakingPerAddress	Public		-
SafeMath	Library			

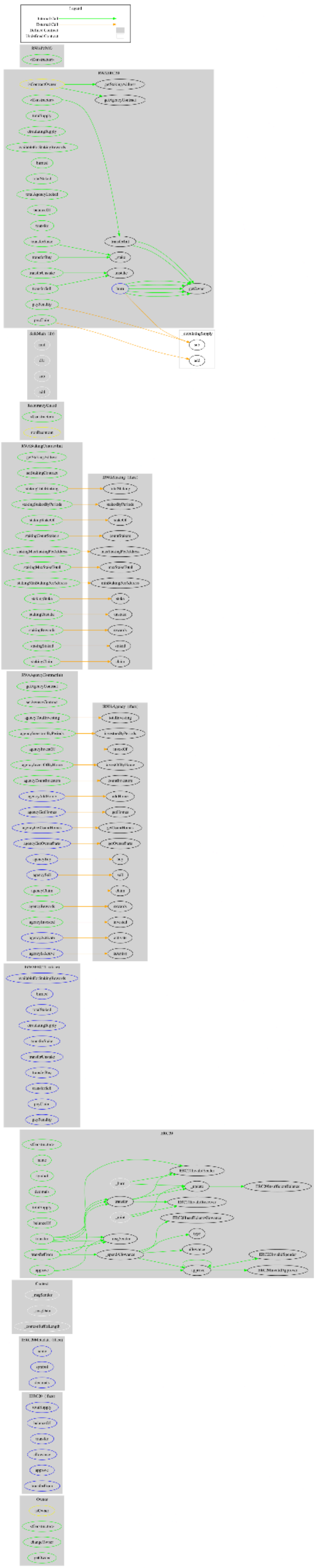
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
RWAERC20	Implementation	ERC20, IRWAERC20, Owner, ReentrancyGuard, RWAStakingContractInit, RWAAgencyContractInit		
		Public	✓	Owner RWAAgencyContractInit RWAStakingContractInit ERC20
	totalSupply	Public		-
	circulatingSupply	Public		-
	availableForStakingRewards	Public		-
	burned	Public		-
	totalStaked	Public		-
	totalAgencyLocked	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	transferInit	Private	✓	nonReentrant
	transferStake	Public	✓	isContractOwner nonReentrant
	transferUnstake	Public	✓	isContractOwner nonReentrant

	transferBuy	Public	✓	isContractOwner nonReentrant
	transferSell	Public	✓	isContractOwner nonReentrant
	payClaim	Public	✓	isContractOwner nonReentrant
	payPenality	Public	✓	isContractOwner nonReentrant
	burn	External	✓	isOwner nonReentrant
	_stake	Private	✓	
	_unstake	Private	✓	
RWAIMMO	Implementation	Owner, RWAERC20		
		Public	✓	RWAERC20

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
RWAERC20
<Constructor>
getAgencyContract
totalSupply
circulatingSupply
burned
transfer
transferBuy
transferSell
payPenalty
_circulatingSupply
sub
add
ReentrancyGuard

Summary

RWA IMMO 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 transferring the user's tokens and burning tokens from any address. If the contract owner abuses the burn functionality, then the users could lose their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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