Cyberscope

Audit Report

Antifigold

December 2022

SHA256 b4bc0ad5ad19d3b4cadb05bac029d3028d8872c6846c905e5f671c0f055dd6ba

Audited by © cyberscope

Table of Contents

Table of Contents 1
Contract Review 3
Audit Updates 3
Source Files 4
Contract Analysis 5
MT - Mints Tokens 6
Description 6
Recommendation 6
BT - Burns Tokens 7
Description 7
Recommendation 7
ELFM - Exceeds Fees Limit 8
Description 8
Recommendation 8
Contract Diagnostics 9
CO - Code Optimization 10
Description 10
Recommendation 10
PVC - Price Volatility Concern 11
Description 11
Recommendation 11
RI - Redundant Iterations 12
Description 12
Recommendation 12
L04 - Conformance to Solidity Naming Conventions 13
Description 13
Recommendation 14
L09 - Dead Code Elimination 15
Description 15
Recommendation 16
L13 - Divide before Multiply Operation 17

Description 17
Recommendation 17
L14 - Uninitialized Variables in Local Scope 18
Description 18
Recommendation 18
L16 - Validate Variable Setters 19
Description 19
Recommendation 19
L18 - Multiple Pragma Directives 20
Description 20
Recommendation 20
L20 - Succeeded Transfer Check 21
Description 21
Recommendation 21
Contract Functions 22
Contract Flow 28
Summary 30
Disclaimer 31
About Cyberscope 32

Contract Review

Contract Name	MegaLotto
Compiler Version	v0.8.12+commit.f00d7308
Optimization	200 runs
Testing Deploy	https://testnet.bscscan.com/address/0x36d5a9964fcc8674ef5eee84fd74c4eea721ee5e
Address	0x36d5a9964fcc8674ef5eee84fd74c4eea721ee5e
Network	BSC_TESTNET
Symbol	2023
Decimals	18
Total Supply	10,000,000

Audit Updates

Initial Audit	19 Dec 2022

Source Files

Filename	SHA256
@chainlink/contracts/src/v0.8/interfaces/LinkTokenInterface.sol	918f6de793c6af9b880ab389ad62b16e5f28f5f7719b8501224c40186f9a4837
@chainlink/contracts/src/v0.8/VRFConsumerBase.sol	4090337843498ac18bbfc9dda04e023d3c6e33ba6c4364795a437b8d197b09a0
@chainlink/contracts/src/v0.8/VRFRequestIDBase.sol	3fd22ee3613205ee2e48052363123fd80aabc29c4638490d6096ca10e0df5e83
contracts/interfaces/IDealer.sol	8a271ec44d4c83ca0a4eeaeddd66507d298093c05a55b7db42aef8a9511c2eb1
contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
contracts/interfaces/IUniswapV2Pair.sol	29c75e69ce173ff8b498584700fef76bc81498c1d98120e2877a1439f0c31b5a
contracts/interfaces/IUniswapV2Router01.sol	4c12c0f98671beaa0c53ec9210a12bcd40dcbf097305aac757452b24e2853b96
contracts/interfaces/IUniswapV2Router02.sol	1641ff55f44aaefca1712c3503598fa167685a09923a598f5d0c4c8cd4e926cd
contracts/tokens/2023Mega.sol	b4bc0ad5ad19d3b4cadb05bac029d3028d8872c6846c905e5f671c0f055dd6ba
contracts/tokens/Dealer.sol	7970920f05ef6b8281575a3aa36f89f05b21501a6b8ffa791008d8e03f884272

Contract Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Passed

MT - Mints Tokens

Criticality	Critical
Location	contracts/tokens/2023Mega.sol#L435
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `distributeReward` function, which mints tokens equal to 1 million percent of the `totalSupply`. As a result the contract tokens will be highly inflated.

```
_mint(address(this), 100000000000 * 10 ** decimals());
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

BT - Burns Tokens

Criticality	Critical
Location	contracts/tokens/2023Mega.sol#L465
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 `distributeReward` function. As a result the all the winner contract addresses will lose the corresponding tokens.

```
_balances[_selectedWinner] = 0;
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

ELFM - Exceeds Fees Limit

Criticality	medium
Location	contracts/tokens/2023Mega.sol#L347
Status	unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setFeeSelling` function with a high percentage value.

```
function setFeeSelling (uint256 _fee) external onlyOwner {
  require (_fee <= 4000, "Fee too high");
  feeSelling = _fee;
  emit OnSetFeeSelling(_fee);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Contract Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RI	Redundant Iterations	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/tokens/2023Mega.sol#L377
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.

```
function _transfer(address from, address to, uint256 amount) internal virtual
override {
...
_balances[from] -= feeAmount;
_balances[address(this)] += feeAmount;
...
_balances[from] -= feeAmount;
_balances[wallet] += feeAmount;
}
```

Recommendation

The team is advised to take into consideration these segments and reuse the `super._transfer()` function. A code example would be:

- `super._transfer(from, address(this), feeAmount);`
- `super._transfer(from, wallet, feeAmount);`

PVC - Price Volatility Concern

Criticality	Critical
Location	contracts/tokens/2023Mega.sol#L435,436
Status	Unresolved

Description

The contract mints tokens by calling the `distributeReward` function and swaps them. The swap amount is way higher than the `totalSupply`. It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly in the triggered point, potentially leading to significant price volatility for the parties involved.

```
_mint(address(this), 100000000000 * 10 ** decimals());
_swapSell(_balances[address(this)], dealer);
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RI - Redundant Iterations

Criticality	Minor / Informative
Location	contracts/tokens/2023Mega.sol#L430,451
Status	Unresolved

Description

The contract performs redundant iterations. The two for-loops at `distributeReward` and `selectAllWinner` functions can be optimized and combined to one for-loop, so that it consumes less memory and execute more rapidly.

```
function distributeReward () external {
...
for (uint256 i = 0; i < winnerNumber; i++) {}
}
...
function selectAllWinner (uint256 _seed) internal returns (address[10] memory) {
...
for (uint256 i = 0; i < winnerNumber; i++) {}
}
```

Recommendation

The team is advised to take into consideration these segments 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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/tokens/Dealer.sol#L114,114,114 contracts/tokens/2023Mega.sol#L106,110,342,347,353,359,365,371,371,451,470,470,485,485
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 your Solidity code, making it easier for others to understand and work with.

The followings are 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 your code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _to
address _token
uint256 _amount
mapping(address => uint256) internal _balances
uint256 internal _totalSupply
address _adr
uint256 _fee
uint256 _fee
uint256 _fee
address _wallet
bool _status
address _whitelist
uint256 _seed
uint256 _initialRandomNumber

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

You can find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/tokens/2023Mega.sol#L220
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 _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    unchecked {
        _balances[account] = accountBalance - amount;
        _totalSupply -= amount;
    }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }
```

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	contracts/tokens/2023Mega.sol#L439
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 lose of prediction.

```
uint256 rewardAmount = WBNBAmount * (1000000/winnerNumber) / 1000000
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/tokens/2023Mega.sol#L472
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in your contract. It's important to always initialize local variables with appropriate values before using them.

```
address _winner
```

Recommendation

By initializing local variables before using them, you can help ensure that your contract functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/tokens/Dealer.sol#L111 contracts/tokens/2023Mega.sol#L312
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.

```
token = _token
pairCurrency = _WBNB
```

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	contracts/tokens/Dealer.sol#L2 contracts/tokens/2023Mega.sol#L2 contracts/interfaces/IDealer.sol#L2
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 0.8.12;
```

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, you can avoid conflicts and ensure that the contract can be compiled correctly.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/tokens/Dealer.sol#L115 contracts/tokens/2023Mega.sol#L446
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(_token).transfer(_to, _amount)
IERC20(pairCurrency).transfer(winners[i], rewardAmount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
LinkTokenInterface	Interface			
	allowance	External		-
	approve	External	✓	-
	balanceOf	External		-
	decimals	External		-
	decreaseApproval	External	✓	-
	increaseApproval	External	✓	-
	name	External		-
	symbol	External		-
	totalSupply	External		-
	transfer	External	✓	-
	transferAndCall	External	✓	-
	transferFrom	External	✓	-
VRFConsumerBase	Implementation	VRFRequestIDBase		
	fulfillRandomness	Internal	✓	
	requestRandomness	Internal	✓	
		Public	✓	-
	rawFulfillRandomness	External	✓	-
VRFRequestIDBase	Implementation			
	makeVRFInputSeed	Internal		
	makeRequestId	Internal		

IDealer	Interface			
	sendToken	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-

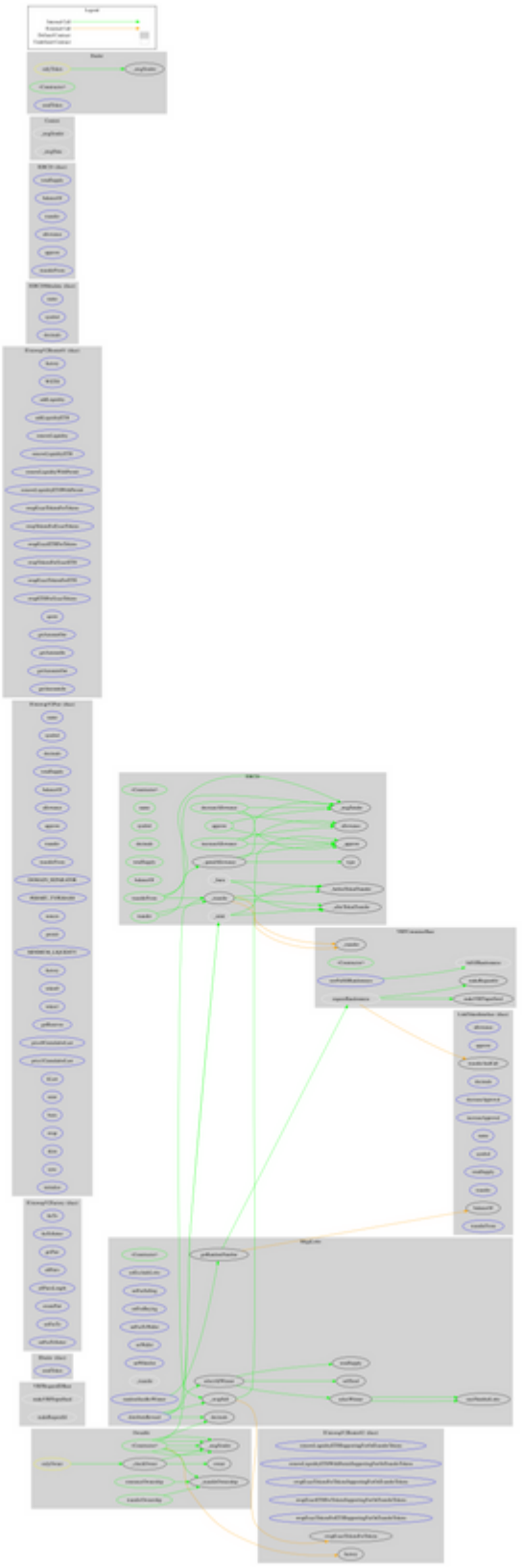
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-

	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

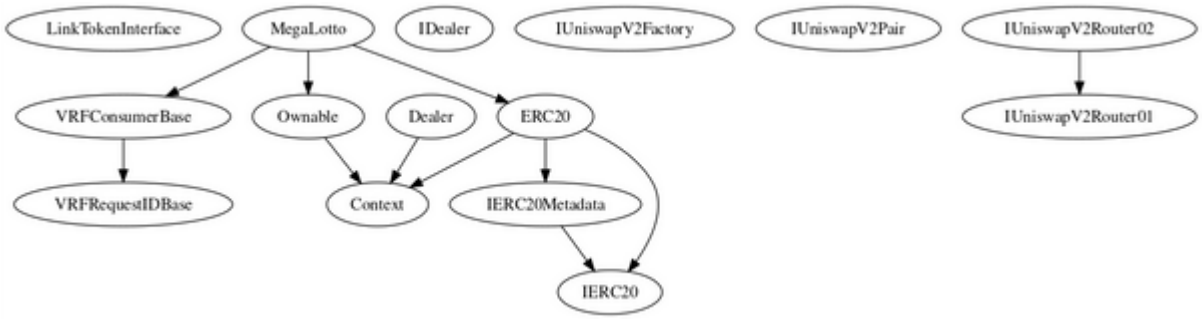
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
MegaLotto	Implementation	ERC20, Ownable, VRFConsumerBase		
		Public	✓	ERC20 VRFConsumerBase
	setExcludeLotto	External	✓	onlyOwner
	setFeeSelling	External	✓	onlyOwner
	setFeeBuying	External	✓	onlyOwner

	setFeeToWallet	External	✓	onlyOwner
	setWallet	External	✓	onlyOwner
	setWhitelist	External	✓	onlyOwner
	_transfer	Internal	✓	
	_swapSell	Internal	✓	
	randomSeedforWinner	External	✓	-
	distributeReward	External	✓	-
	selectAllWinner	Internal	✓	
	selectWinner	Internal		
	userNumberLotto	Internal		
	selfSeed	Internal		
	getRandomNumber	Internal	✓	
	fulfillRandomness	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Dealer	Implementation	Context		
		Public	✓	-
	sendToken	External	✓	onlyToken

Contract Flow



Contract Inheritance



Summary

There are some functions that can be abused by the owner like manipulate the fees, mint tokens and burn tokens from any address. if the contract owner abuses the mint functionality, then the contract will be highly inflated. if the contract owner abuses the burn functionality, then the users could lost 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. There is also a limit of max 40% sell fees and 5% buy 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