Cyberscope

Audit Report

MuziKoin

February 2024

Network	BSC
Address	0xCe379bAD99f287E1eacB1bF80F3670011D2B2981
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RED	Redundant Event Declaration	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	ULTW	Transfers Liquidity to Team Wallet	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L05	Unused State Variable	Unresolved
	L09	Dead Code Elimination	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved
	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	MuziKoin
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xce379bad99f287e1eacb1bf80f3670011d2b2981
Address	0xce379bad99f287e1eacb1bf80f3670011d2b2981
Network	BSC
Symbol	MZK
Decimals	18
Total Supply	900,000,000
Badge Eligibility	Yes

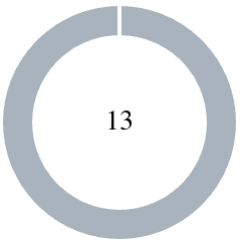
Audit Updates

Initial Audit	05 Feb 2024

Source Files

Filename	SHA256
MuziKoin.sol	3367464fd863bca7e233fdeb1baaef7dceba4780a3cef8e1fad804512677f1ec

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MuziKoin.sol#L842
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.

```
cxWallet
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	MuziKoin.sol#L1021
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 {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = WETH;
    _approve(address(this), address(uniswapV2Router),
tokenAmount);

    // make the swap

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this), // The contract
        block.timestamp
    );

    emit SwapTokensForETH(tokenAmount, path);
}
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	MuziKoin.sol#L1063
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

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 at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setTokensToSwap(
    uint256 _minimumTokensBeforeSwap
) external onlyOwner {
    require(
        _minimumTokensBeforeSwap >= 100 ether,
        "You need to enter more than 100 tokens."
    );
    minimumTokensBeforeSwap = _minimumTokensBeforeSwap;
    emit UpdateTokensToSwap(minimumTokensBeforeSwap);
}

...

bool overMinimumTokenBalance = contractTokenBalance >=
minimumTokensBeforeSwap;
uint fee = 0;
    //if any account belongs to _isExcludedFromFees account
then remove the fee
if (
!inSwapAndLiquify &&
    from != uniswapV2Pair &&
    overMinimumTokenBalance &&
    swapAndLiquifyEnabled
) {
    swapAndLiquify();
}
```

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 exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	MuziKoin.sol#L784,785,793,795
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.

The following events are declared and not being used in the contract. As a result, they are redundant.

```
event Log(string, uint256);
event AuditLog(string, address);
event RewardLiquidityProviders(uint256 tokenAmount);
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiqudity
    );
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	MuziKoin.sol#L1063,1087,1093
Status	Unresolved

Description

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

```
function setTokensToSwap(
    uint256 _minimumTokensBeforeSwap
) external onlyOwner {
    require(
        _minimumTokensBeforeSwap >= 100 ether,
        "You need to enter more than 100 tokens."
    );
    minimumTokensBeforeSwap = _minimumTokensBeforeSwap;
    emit UpdateTokensToSwap(minimumTokensBeforeSwap);
}

function setBuyFee(uint256 _buyFee) external onlyOwner {
    require(_buyFee <= 5, "Buy Fee cannot be more than 5%");
    buyFee = _buyFee;
    emit UpdateBuyFee(buyFee);
}

function setSellFee(uint256 _sellFee) external onlyOwner {
    require(_sellFee <= 5, "Sell Fee cannot be more than 5%");
    sellFee = _sellFee;
    emit UpdateSellFee(sellFee);
}
```

Recommendation

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

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	MuziKoin.sol#L1010
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the `marketingWallet`. The owner may take advantage of it by calling the `swapAndLiquify` method.

```
function swapAndLiquify() public lockTheSwap {
    uint256 totalTokens = balanceOf(address(this));
    swapTokensForEth(totalTokens);
    uint ethBalance = address(this).balance;

    transferToAddressETH(marketingWallet, ethBalance);

    marketingTokensCollected = 0;
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. 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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MuziKoin.sol#L802,803
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private _tTotal = 900_000_000 ether
uint256 private _tFeeTotal
```

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	MuziKoin.sol#L499,503,531,571,791,792,821,903,1064,1074,1081,1087,1091,1144,1145
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);
event recoveredETH(uint256);
event recoveredTokens(uint256);
address private immutable WETH
address _owner
uint256 _minimumTokensBeforeSwap
bool _enabled
address _marketingWallet
uint256 _buyFee
uint256 _sellFee
address _tokenAddress

...
```

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	MuziKoin.sol#L803
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.

```
uint256 private _tFeeTotal
```

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	MuziKoin.sol#L79,103,134,153,172,192,220,238,259,277,298,322,334,1117
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) {
        // This method relies on
extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is
only stored at the end
        // of the constructor execution.

        return account.code.length > 0;
    }

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MuziKoin.sol#L834
Status	Unresolved

Description

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

```
address currentRouter
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MuziKoin.sol#L899,957
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	MuziKoin.sol#L342
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

Recommendation

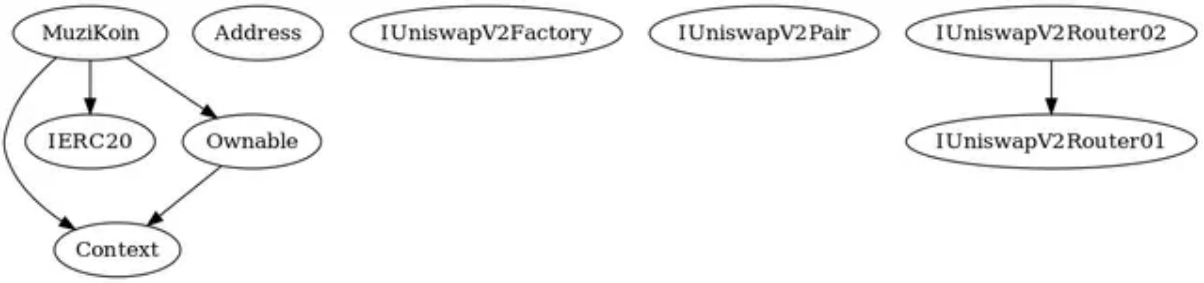
It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Functions Analysis

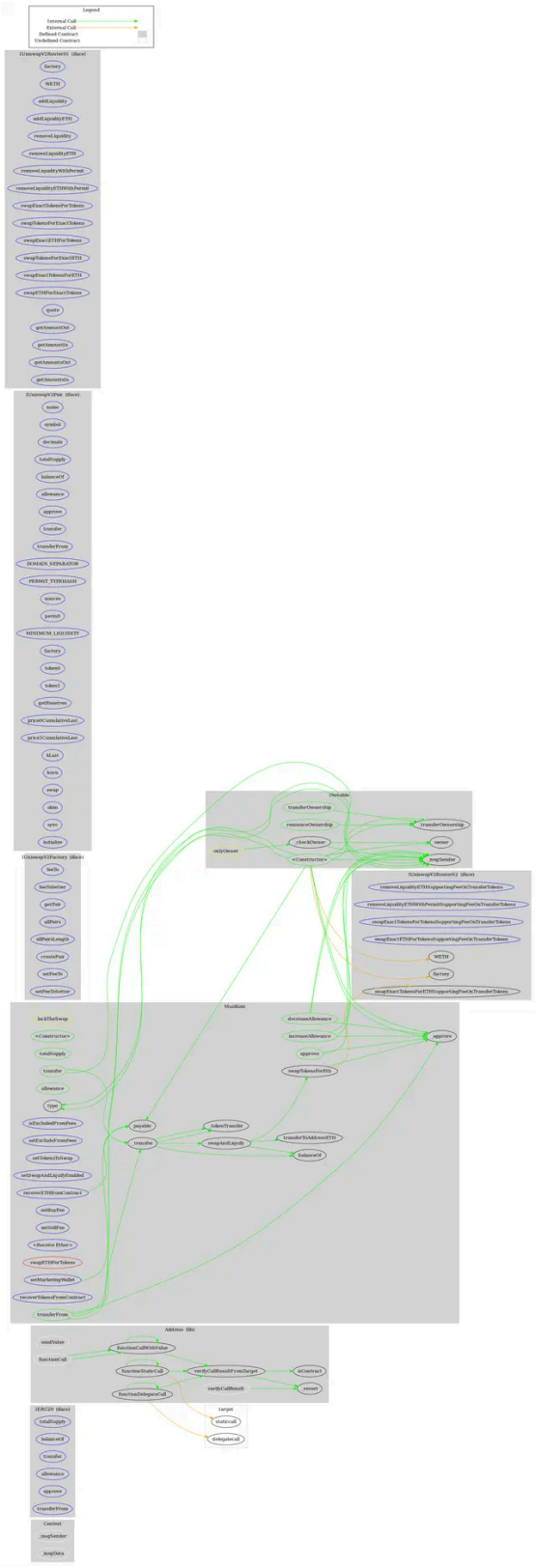
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MuziKoin	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Public	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	_tokenTransfer	Private	✓	
	isExcludedFromFees	External		-
	setExcludeFromFees	External	✓	onlyOwner
	setTokensToSwap	External	✓	onlyOwner

	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setBuyFee	External	✓	onlyOwner
	setSellFee	External	✓	onlyOwner
	transferToAddressETH	Private	✓	
		External	Payable	-
	swapETHForTokens	Private	✓	
	recoverETHfromContract	External	✓	onlyOwner
	recoverTokensFromContract	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
IUniswapV2Pair
name
symbol
decimals
totalSupply
balanceOf
allowance
approve
transfer
transferFrom
DOMAIN_SEPARATOR
PERMIT_TYPEHASH
nonces
permit
MINIMUM_LIQUIDITY
factory
token0
token1
getReserves
price0CumulativeLast
price1CumulativeLast
kLast
burn
swap
skim
sync
initialize
IUniswapV2Factory
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
Ownable
transferOwnership
renounceOwnership
checkOwner
<Constructor>
owner
_msgSender
onlyOwner
IUniswapV2Router02
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
WETH
factory
swapExactTokensForETHSupportingFeeOnTransferTokens
MuziKoin
lockTheSwap
<Constructor>
totalSupply
transfer
allowance
_type
isExcludedFromFees
setIsExcludedFromFees
setTokensToSwap
setSwapAndLiquifyEnabled
recoverETHfromContract
setBuyFee
setSellFee
<Receive Ether>
swapETHForTokens
setMarketingWallet
recoverTokensFromContract
transferFrom
decreaseAllowance
increaseAllowance
approve
swapTokensForEth
payable
_tokenTransfer
transfer
swapAndLiquify
transferToAddressETH
balanceOf
Address
sendValue
functionCallWithValue
functionCall
functionStaticCall
verifyCallResultFromTarget
isContract
functionDelegateCall
verifyCallResult
revert
target
staticcall
delegatecall
IERC20
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

Summary

MuziKoin 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 5% 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