Cyberscope

Audit Report

Blocklink

May 2023

BlockATM	56550e1ec324bc2c57703c1b05c700c5ef234302c740083f22c08013ef173da1
BlockATMCustomer	bbab73bdf5d206668343b3dd5ef3b713bc283eda30e782b5cd1a4f48c8bfe648
BlockOrder	89d522915f80e39255dff3cdb10ba2d84bb0a04ca28ed29c6d37d0fa81daf004
BlockRecharge	2ab1ca3f03b6477cb43bae6f5240913d01d0e0ebb52fcac1108f344083d7e9f7

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	24 May 2023

Source Files

Filename	SHA256
BlockATM.sol	56550e1ec324bc2c57703c1b05c700c5ef234302c740083f22c08013ef173da1
BlockATMCustomer.sol	bbab73bdf5d206668343b3dd5ef3b713bc283eda30e782b5cd1a4f48c8bfe648
BlockOrder.sol	89d522915f80e39255dff3cdb10ba2d84bb0a04ca28ed29c6d37d0fa81daf004
BlockRecharge.sol	2ab1ca3f03b6477cb43bae6f5240913d01d0e0ebb52fcac1108f344083d7e9f7

Introduction

The Blocklink ecosystem consists of four distinct contracts: BlockATM, BlockATMCustomer, BlockOrder, and BlockRecharge. The BlockATM contract serves as an automated teller machine for cryptocurrencies, enabling users to transfer ERC20 tokens and Ether (ETH) to a designated withdrawal address. The BlockATMCustomer contract is designed for customers of the BlockATM system, allowing them to transfer ERC20 tokens and subsequently withdraw them to specified addresses. BlockOrder facilitates the transfer of ERC20 tokens associated with orders or specific transactions to a designated withdrawal address. Lastly, BlockRecharge offers the same functionality as the BlockOrder contract. These contracts provide various features and functionality related to token transfers and withdrawals, with the contract owners having control over withdrawal addresses, supported tokens, and other related settings.

Roles

BlockATM

Owner

The owner has authority over the following functions:

- `function modifyWithdrawAddress(address payable _address)`
- `function addCoinList(address _address)`

User

The user can interact with the following functions:

- `function transferToken(address token,uint256 amount,string memory orderId)`
- `function transferETH(string memory orderId)`
- `function getWithdrawAddress()`
- `function getCoinList(address _address)`

BlockATMCustomer

Owner

The owner has authority over the following functions:

- `function withdrawToken(uint256 amount,address withdrawAddress)`
- `function burn()`

User

The user can interact with the following functions:

- `function transferToken(uint256 amount,string memory orderId)`
- `function getTokenAddress()`
- `function getActiveFlag()`
- `function getWithdrawAddressList()`
- `function checkWithdrawAddress(address withdrawAddress)`
- `function checkOnwerAddress(address ownerAddress)`
- `function getOwnerAddressList()`

BlockOrder

Owner

The owner has authority over the following functions:

- `function modifyWithdrawAddress(address payable _address)`
- `function addCoinList(address _address)`
- `function closeCoinList(address _address)`

User

The user can interact with the following functions:

- `function transferToken(address token,uint256 amount,string memory orderId)`
- `function getWithdrawAddress()`
- `function getCoinList(address _address)`

BlockRecharge

Owner

The owner has authority over the following functions:

- `function modifyWithdrawAddress(address payable _address)`
- `function addCoinList(address _address)`
- `function closeCoinList(address _address)`

User

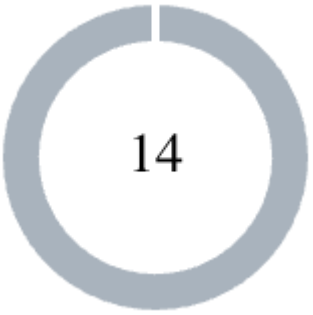
The user can interact with the following functions:

- `function transferToken(address token,uint256 amount,string memory orderId)`
- `function getWithdrawAddress()`
- `function getCoinList(address _address)`

Test Deployments

Contract	Explorer
BlockATM	https://testnet.bscscan.com/address/0xdae8eb401b0683747ea7a515867b07b6839282a0
BlockATMCustomer	https://testnet.bscscan.com/address/0x4b43f9d8f60ab34c88ba0eda9e656663508954c0
BlockOrder	https://testnet.bscscan.com/address/0xcA7fCc252C678E28D187ad0ab9Fea056C5e6A43E
BlockRecharge	https://testnet.bscscan.com/address/0x5D490EF16af2a3B965f12067f3fAB5e97aAc6268

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

Findings

Severity	Code	Description	Status
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	POID	Potential Order Id Duplications	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	CI	Contracts Inconsistency	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	DSM	Data Structure Misuse	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	CIO	Contract Interface Optimization	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

The contract emits an event that describes the transfer operation. The argument 'amount' describes the amount of tokens that were transferred to the contract. If the transfer is taxed by the token, then the event will be misleading since the actual amount will be less than the logger's amount.

```
bool success = erc20.transferFrom(msg.sender, address(this),
amount);
require(success, "transfer token failed");
emit TransferToken(msg.sender, address(this), tokenAddress,
amount,orderId);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

POID - Potential Order Id Duplications

Criticality	Minor / Informative
Location	BlockATM.solBlockATMCustomer.solBlockOrder.solBlockRecharge.sol
Status	Unresolved

Description

The contract uses an identifier for every transfer operation that is called `orderId`. The `orderId` seems to help the users from identifying unique transactions and track the proper operation of the ecosystem. The identifiers are usually unique. The contract does not provide any guarantee that the `orderId` is unique. As a result, a transfer operation can be triggered more than once for a given `orderId`.

Recommendation

The team is advised to implement a mechanism in order to avoid potential duplications of the `orderId` execution.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	BlockATMCustomer.sol#L104
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The balance of contract is the accumulated amount of all the users' deposits. The owner may take advantage of it by adding only the owner's address to the `ownerMap` variable and then calling the `withdrawToken` function.

```
modifier onlyOwner() {
    require(ownerMap[msg.sender] == 1, "Not the owner");
    _;
}

function withdrawToken(uint256 amount,address withdrawAddress)
public onlyOwner returns (bool) {}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

CI - Contracts Inconsistency

Criticality	Minor / Informative
Location	BlockATM.solBlockATMCustomer.solBlockOrder.solBlockRecharge.sol
Status	Unresolved

Description

The Blocklink ecosystem consists of four contracts. These contracts are meant to simulate the behaviour of an ATM. However, each contract is completely independent from one another. There is no connection link to ensure the credibility and predictability of the transactions and the ecosystem overall. As a result, this lack of connection between the contracts may lead to potential inconsistencies in the system, as well as the users' confidence of credibility, making it difficult to maintain a uniform and predictable behavior.

Recommendation

The team is advised to establish a connection or link between the four independent contracts. This connection will help ensure the credibility, predictability, and uniform behavior of the transactions within the ecosystem. Some recommendations to consider would be:

- Implement a communication mechanism between the four contracts to establish a link and enable them to share relevant information.
- Create shared state variables that can be accessed and updated by all contracts as necessary.
- Implement appropriate validation and consistency checks across the contracts to ensure that the data being shared is accurate and consistent.
- When implementing inter-contract communication, ensure that proper security measures are in place to prevent unauthorized access or manipulation of shared data.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	BlockATM.sol#L68,76BlockOrder.sol#L60,68,72BlockRecharge.sol#L60,68,72
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 contract modifier the state of some variables without checking if their current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
withdrawAddress = _address;
coinList[_address] = 1;
coinList[_address] = 0;
```

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.

DSM - Data Structure Misuse

Criticality	Minor / Informative
Location	BlockATM.sol#L25BlockATMCustomer.sol#L23,25BlockOrder.sol#L26BlockRecharge.sol#L26
Status	Unresolved

Description

The contract utilizes three mappings, namely `coinList`, `withdrawMap`, and `ownerMap`, which map addresses to unsigned integers. However, these unsigned integers only hold values of either 1 or 0. Hence, it is not necessary for these variables to store a uint256.

```
mapping(address => uint) private coinList;
mapping(address => uint) private withdrawMap;
mapping(address => uint) private ownerMap;
```

Recommendation

Since the contract stores only 0 and 1 integers, then a boolean data type could be used. The false value could simulate the number zero and the true value could simulate the number one.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	BlockATMCustomer.sol#L104
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

Intuitively a `burn()` method is used to burn tokens. The contract is using the `burn()` method in order to disable the execution of the `transferToken()` method forever. As a result, the method name is misleading.

```
function burn() public onlyOwner {
    require(activeFlag, "The contract has already burned");
    // burn contract
    IERC20 erc20 = IERC20(tokenAddress);
    uint256 balance = erc20.balanceOf(address(this));
    // check balance
    require(balance == 0, "Please withdraw the balance");
    activeFlag = false;
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

CIO - Contract Interface Optimization

Criticality	Minor / Informative
Location	BlockATM.sol#L4BlockATMCustomer.sol#L4BlockOrder.sol#L4BlockRecharge.sol#L4
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.

All of the contracts declare the same IERC20 interface which is used for ERC20 tokens interactions. Since the interface is the same for all the contracts, it could be reused instead of declaring it on every contract.

```
interface IERC20 {

    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns
(uint256);

    function transfer(address recipient, uint256 amount) external
returns (bool);

    function allowance(address owner, address spender) external
view returns (uint256);

    function approve(address spender, uint256 amount) external
returns (bool);

    function transferFrom(address sender, address recipient,
uint256 amount) external returns (bool);
}
```

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.

A recommended approach would be to extract the interface in a separate file and import that file on every contract.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	BlockATMCustomer.sol#L74
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The contract does not validate if the contract's tokens are greater than or equal to the given amount.

```
bool success = erc20.transfer(withdrawAddress, amount);
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	BlockATM.sol#L67,75BlockOrder.sol#L59,67,71BlockRecharge.sol#L59,67,71BlockATMCustomer.sol#L104
Status	Unresolved

Description

Detected missing events for critical access control parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
function modifyWithdrawAddress(address payable _address) public
onlyOwner {
    withdrawAddress = _address;
}
function addCoinList(address _address) public onlyOwner {
    coinList[_address] = 1;
}
...
```

Recommendation

The team is advised to emit an event for all critical parameter changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	BlockATM.sol#L30BlockATMCustomer.sol#L34,38,42BlockOrder.sol#L31 BlockRecharge.sol#L31
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
_owner
tokenAddress
_withdrawList
_ownerList
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BlockRecharge.sol#L59,63,67,71BlockOrder.sol#L59,63,67,71BlockATM.sol#L67,71,75
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.

```
address payable _address
address _address
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	BlockRecharge.sol#L30,60BlockOrder.sol#L30,60BlockATMCustomer.sol#L34BlockATM.sol#L29,68
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.

```
withdrawAddress = _withdrawAddress
withdrawAddress = _address
tokenAddress = _tokenAddress
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BlockRecharge.sol#L2BlockOrder.sol#L2BlockATMCustomer.sol#L2BlockATM.sol#L2
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 ^0.8.0;
```

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
BlockATM	Implementation			
		Public	✓	-
	transferToken	Public	✓	-
	transferETH	Public	Payable	-
	getWithdrawAddress	Public		-
	modifyWithdrawAddress	Public	✓	onlyOwner
	getCoinList	Public		-
	addCoinList	Public	✓	onlyOwner
IERC20	Interface			

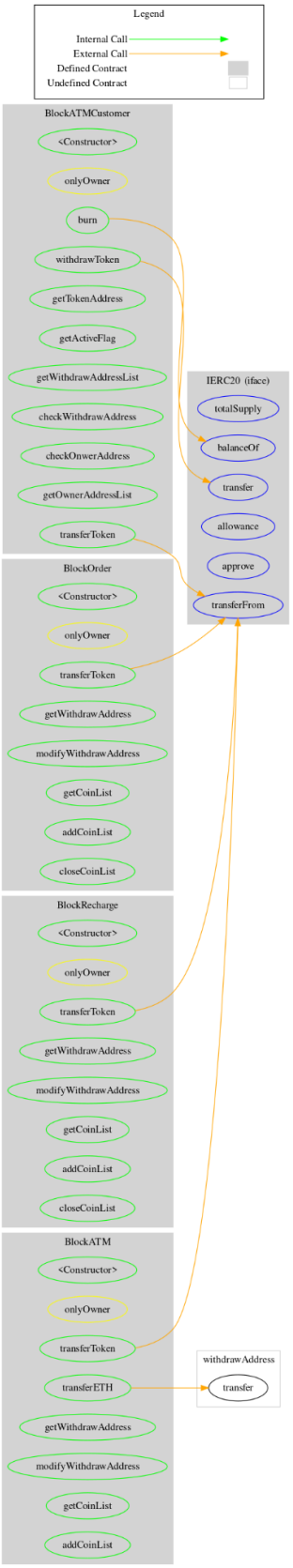
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
BlockATMCustomer	Implementation			
		Public	✓	-
	transferToken	Public	✓	-
	withdrawToken	Public	✓	onlyOwner
	getTokenAddress	Public		-
	getActiveFlag	Public		-
	getWithdrawAddressList	Public		-
	checkWithdrawAddress	Public		-
	checkOnwerAddress	Public		-
	getOwnerAddressList	Public		-
	burn	Public	✓	onlyOwner
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
BlockOrder	Implementation			
		Public	✓	-
	transferToken	Public	✓	-
	getWithdrawAddress	Public		-
	modifyWithdrawAddress	Public	✓	onlyOwner
	getCoinList	Public		-
	addCoinList	Public	✓	onlyOwner
	closeCoinList	Public	✓	onlyOwner
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
BlockRecharge	Implementation			
		Public	✓	-

	transferToken	Public	✓	-
	getWithdrawAddress	Public		-
	modifyWithdrawAddress	Public	✓	onlyOwner
	getCoinList	Public		-
	addCoinList	Public	✓	onlyOwner
	closeCoinList	Public	✓	onlyOwner

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BlockATMCustomer
<Constructor>
onlyOwner
burn
withdrawToken
getTokenAddress
getActiveFlag
getWithdrawAddressList
checkWithdrawAddress
checkOnwerAddress
getOwnerAddressList
transferToken
BlockOrder
<Constructor>
onlyOwner
transferToken
getWithdrawAddress
modifyWithdrawAddress
getCoinList
addCoinList
closeCoinList
BlockRecharge
<Constructor>
onlyOwner
transferToken
getWithdrawAddress
modifyWithdrawAddress
getCoinList
addCoinList
closeCoinList
BlockATM
<Constructor>
onlyOwner
transferToken
transferETH
getWithdrawAddress
modifyWithdrawAddress
getCoinList
addCoinList
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
withdrawAddress
transfer

Summary

Blocklink contract implements a financial mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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