Cyberscope

Audit Report

Barin Mineral Token

June 2025

File	SHA256
BarinToken	9708f423745d99492822331d38f95fe86b5c86c12e174ef4efdf4cead10c6b5e
BarinVesting	12ca1e3981efbbde6a1a337696f364c47e5987a38255499c1fbd410791ef9b46
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit	29 May 2025
Corrected Phase 2	12 Jun 2025

Source Files

Filename	SHA256
BarinToken.sol	9708f423745d99492822331d38f95fe86b5c86c12e174ef4efdf4cead10c6b5e
BarinVesting.sol	12ca1e3981efbbde6a1a337696f364c47e5987a38255499c1fbd410791ef9b46

Overview

Barin Mineral Token (BARIN)

The `Barin` contract represents the Barin Mineral Token (BARIN), an ERC20 token that incorporates support for EIP-2612, enabling gasless approvals via the `ERC20Permit` extension. The token follows the standard ERC20 behavior and allows for efficient, secure transfers and approvals. The contract is built on OpenZeppelin's ERC20 and ERC20Permit contracts to provide industry-standard functionality while enabling users to approve token transfers without incurring gas fees for approval transactions.

Gasless Approvals with EIP-2612

By inheriting from `ERC20Permit`, the `Barin` contract supports the EIP-2612 standard for gasless approvals. This allows users to approve token transfers without needing to send a transaction on the blockchain, saving gas fees for approval actions. The `ERC20Permit` extension uses a signature-based method for granting approval, enabling users to sign off-chain messages to approve spending by other addresses.

Token Minting

Upon deployment, the contract mints a total supply of 1 billion `BARIN` tokens (1,000,000,000) and assigns the entire supply to the contract deployer's address. This ensures that the deployer has full control of the initial token balance.

The token supply is minted with the following formula:

- **Initial Mint**: `1,000,000,000 * 10^decimals()` tokens.

Functionality

Inherited Functionality

The contract inherits and extends the functionality of the following OpenZeppelin contracts:

- **`ERC20`**: Implements the ERC20 token standard, allowing for basic token functionality such as transferring, approving, and querying balances and allowances.

- `ERC20Permit`: Adds support for EIP-2612, enabling gasless approvals via permit signatures.

BarinVesting Contract

The `BarinVesting` contract is designed to manage multiple vesting schedules for the BARIN token. It provides functionality for creating and managing vesting schedules, releasing vested tokens, and revoking unvested portions, if allowed. The contract is flexible and enables the owner to configure various parameters. It ensures the gradual release of tokens according to predefined schedules. The contract utilizes OpenZeppelin's `Ownable` and `ReentrancyGuard` to enhance security and prevent reentrancy attacks.

Vesting Schedule Creation

The `createVestingSchedule` function allows the contract owner to create a single vesting schedule for a beneficiary. This function requires specifying the total amount of tokens to be vested, the cliff duration, the vesting duration, and whether the schedule is revocable. Additionally, the `batchCreateSchedules` function allows for batch creation of multiple vesting schedules for several beneficiaries in a single transaction, improving gas efficiency.

Each vesting schedule consists of the following key elements:

- **Cliff Duration**: The period before any tokens are released.
- **Vesting Duration**: The total period over which tokens are gradually released.
- **Revocability**: Determines whether the contract owner can revoke the vesting schedule and reclaim unvested tokens.

When a vesting schedule is created, an event is emitted, providing transparency and tracking.

Token Release

Beneficiaries can claim their vested tokens after the cliff duration has passed. The `release` function calculates how much can be claimed based on the elapsed time since the vesting started. It ensures that only the beneficiary or the contract owner can release the tokens. If no tokens are available for release, the contract throws an error.

The `batchRelease` function is an efficient method that allows the contract owner to release tokens for multiple beneficiaries in one transaction.

Revocation of Vesting Schedules

The `revoke` function allows the contract owner to revoke a vesting schedule, provided that it is marked as revocable. When a schedule is revoked, the unvested tokens are returned to the contract owner. Once a schedule is revoked, it becomes non-revocable, and no further action can be taken. The contract ensures that only unvested tokens are returned to the owner.

Token Vesting Calculation

The vesting process follows a linear model after the cliff period. Tokens are gradually released over the total vesting duration. The amount of tokens released at any given time is proportional to the elapsed time. If the cliff period hasn’t passed, no tokens are released. Once the cliff period is over, the tokens are released linearly over the remaining vesting duration.

Emergency Withdrawals

The contract owner has the ability to withdraw tokens that were accidentally sent to the contract address. This ensures that the owner retains control over the contract's balance in unforeseen situations.

Functionality

Events

The contract emits several events to facilitate transparency and enable off-chain tracking of key operations:

- **VestingScheduleCreated**: Emitted when a new vesting schedule is successfully created. Includes the schedule ID, beneficiary address, and total allocated token amount.
- **TokensReleased**: Emitted when tokens are released to a beneficiary. Contains the schedule ID, beneficiary address, and amount released.

- **VestingRevoked**: Emitted when a revocable vesting schedule is revoked by the owner. Includes the schedule ID and the amount of unvested tokens returned to the owner.
- **EmergencyWithdraw**: Emitted when the owner withdraws tokens from the contract using the emergency withdrawal function. Indicates the token address and withdrawn amount.

Errors

The contract defines custom error types to ensure secure and predictable behaviour:

- **InvalidSchedule**: Thrown when a vesting schedule is misconfigured (e.g., zero address, zero amount, or mismatched input array lengths).
- **CliffLongerThanDuration**: Thrown if the specified cliff period exceeds the total vesting duration.
- **NothingToRelease**: Thrown when a release is attempted but no vested tokens are available for withdrawal.
- **NotBeneficiary**: Thrown if a caller who is neither the beneficiary nor the contract owner attempts to release tokens.
- **ScheduleNotRevocable**: Thrown when a non-revocable schedule is attempted to be revoked.
- **InsufficientBalance**: Thrown if the contract does not have enough available tokens to allocate a new vesting schedule.
- **Underflow**: Thrown if the stored withdrawn amount exceeds the vested amount (indicating a critical arithmetic error).

Constructor

The constructor initializes the contract with two parameters:

- `_token` : The address of the BARIN token contract.
- `_vestingStart` : The start time for vesting, which defaults to the current block timestamp if not provided.

Functions

The contract provides several functions to manage vesting schedules:

- `createVestingSchedule` : Creates a vesting schedule for a single beneficiary.

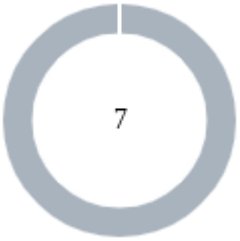
- `batchCreateSchedules` : Allows the contract owner to create multiple vesting schedules in a single transaction.
- `release` : Releases vested tokens for a specific schedule.
- `batchRelease` : Allows the contract owner to release tokens for multiple schedules at once.
- `revoke` : Revokes a vesting schedule and returns unvested tokens to the owner.
- `emergencyWithdraw` : Enables the owner to recover tokens that were accidentally sent to the contract.

View Functions

The contract also provides several view functions to retrieve schedule details:

- `getSchedule` : Returns the details of a specific vesting schedule.
- `releasableAmount` : Returns the amount of tokens that can be released from a specific schedule.
- `vestedAmount` : Returns the total vested amount for a specific schedule.
- `getBeneficiarySchedules` : Returns all vesting schedules for a specific beneficiary.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CCR	Contract Centralization Risk	Unresolved
●	EFO	External Function Optimization	Unresolved
●	MEM	Misleading Error Message	Unresolved
●	MC	Missing Check	Unresolved
●	RBL	Redundant Batch Logic	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	TCT	Transfer Contract Tokens	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	BarinVesting.sol#L66,79,156,199,267
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function createVestingSchedule(address beneficiary, uint256 totalAmount,
uint256 cliffDuration, uint256 vestingDuration, bool revocable) external
onlyOwner
function batchCreateSchedules(address[] calldata beneficiaries, uint256[]
calldata amounts, uint256[] calldata cliffDurations, uint256[] calldata
vestingDurations, bool[] calldata revocables) external onlyOwner
function release(bytes32 scheduleId) external nonReentrant {
    if (schedule.beneficiary != msg.sender && msg.sender != owner()) {
        revert NotBeneficiary();
    }
}
function batchRelease(bytes32[] calldata scheduleIds) external
function revoke(bytes32 scheduleId) external onlyOwner
function emergencyWithdraw(address tokenAddress, uint256 amount) external
onlyOwner
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

EFO - External Function Optimization

Criticality	Minor / Informative
Location	BarinVesting.sol#L156
Status	Unresolved

Description

`batchRelease` calls the `release` function multiple times. The keyword `this` is used to call it since the function is declared as external. The team could instead declare the `release` function as public.

```
function release(bytes32 scheduleId) external nonReentrant
...
this.release(scheduleIds[i]);
```

Recommendation

It is recommended that the team declares the `release` function as public to increase the `batchRelease` s` efficiency.

MEM - Misleading Error Message

Criticality	Minor / Informative
Location	BarinVesting.sol#L159
Status	Unresolved

Description

Errors can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some error names that are too generic or do not clearly convey the underneath functionality.

Specifically, in `release` function the revert message only mentions that the `msg.sender` is not the beneficiary but does not mention the owner.

```
if (schedule.beneficiary != msg.sender && msg.sender !=
owner()) {
    revert NotBeneficiary();
}
```

Recommendation

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

MC - Missing Check

Criticality	Minor / Informative
Location	BarinVesting.sol#L79,156,199
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.

In `release` , `batchRelease` , `revoke` checks are missing to ensure that `scheduleIds` are valid and exist.

```
function release(bytes32 scheduleId) external nonReentrant
function batchRelease(bytes32[] calldata scheduleIds) external
function revoke(bytes32 scheduleId) external onlyOwner
```

Recommendation

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

RBL - Redundant Batch Logic

Criticality	Minor / Informative
Location	BarinVesting.sol#L185
Status	Unresolved

Description

The contract is performing redundant operations within the `batchRelease` function by manually re-executing logic that is already encapsulated within the `release` function, such as recalculating the vested amount and comparing it to the withdrawn amount. This results in duplicated checks and unnecessary computation before ultimately calling `release`, which performs the same validations again. These overlapping operations introduce code inefficiency, make maintenance harder, and may lead to inconsistencies if the logic in `release` is ever updated without updating `batchRelease` accordingly.

```
  function release(bytes32 scheduleId) external nonReentrant {
        VestingSchedule storage schedule = _schedules[scheduleId];

        if (schedule.beneficiary != msg.sender && msg.sender !=
owner()) {
            revert NotBeneficiary();
        }

        uint256 vested = _vestedAmount(scheduleId);
        uint256 withdrawn = schedule.withdrawnAmount;
        if (vested < withdrawn) {
            revert Underflow();
        }
        ...
    }

    function batchRelease(bytes32[] calldata scheduleIds) external {
        for (uint256 i = 0; i < scheduleIds.length; i++) {
            bytes32 sid = scheduleIds[i];
            uint256 vested = _vestedAmount(sid);
            uint256 withdrawn = _schedules[sid].withdrawnAmount;
            if (vested > withdrawn) {
                this.release(sid);
            }
        }
    }
```

Recommendation

It is recommended to simplify the `batchRelease` function by removing the redundant logic and directly iterating over the input schedule IDs to call the existing `release` function. This ensures that all release conditions are consistently enforced in one location and avoids the risk of code duplication or divergence.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	BarinVesting.sol#L242
Status	Unresolved

Description

The contract is designed to be provided the tokens externally. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks. Additionally tokens from the contract can be withdrawn from the owner which may result in beneficiaries not be able to release their tokens.

```
function createVestingSchedule
...
function batchCreateSchedules
...
function release
...
function batchRelease
...
function emergencyWithdraw(address tokenAddress, uint256 amount) external
onlyOwner {
    IERC20(tokenAddress).safeTransfer(owner(), amount);
    emit EmergencyWithdraw(tokenAddress, amount);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution could be to transfer the tokens in the contract during its initialization or during the creation of the vesting schedules. Additionally, the team could consider the amount of tokens vested in the `emergencyWithdraw` function.

TCT - Transfer Contract Tokens

Criticality	Minor / Informative
Location	BarinVesting.sol#L267
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `emergencyWithdraw` function.

```
    function emergencyWithdraw(address tokenAddress, uint256 amount)
external onlyOwner {
        IERC20(tokenAddress).safeTransfer(owner(), amount);
        emit EmergencyWithdraw(tokenAddress, amount);
    }
```

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:

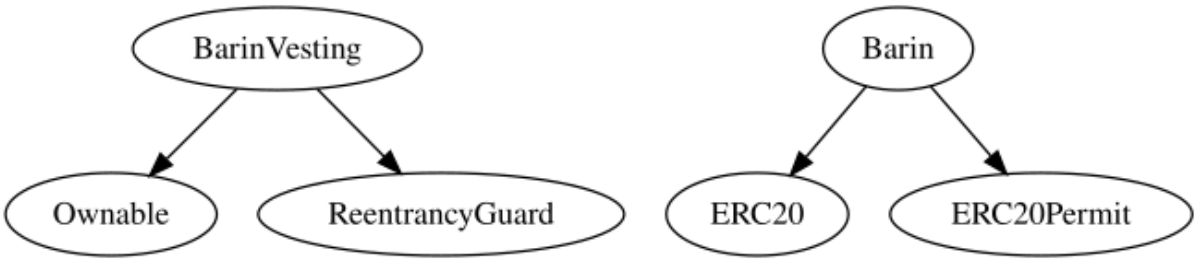
- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

Functions Analysis

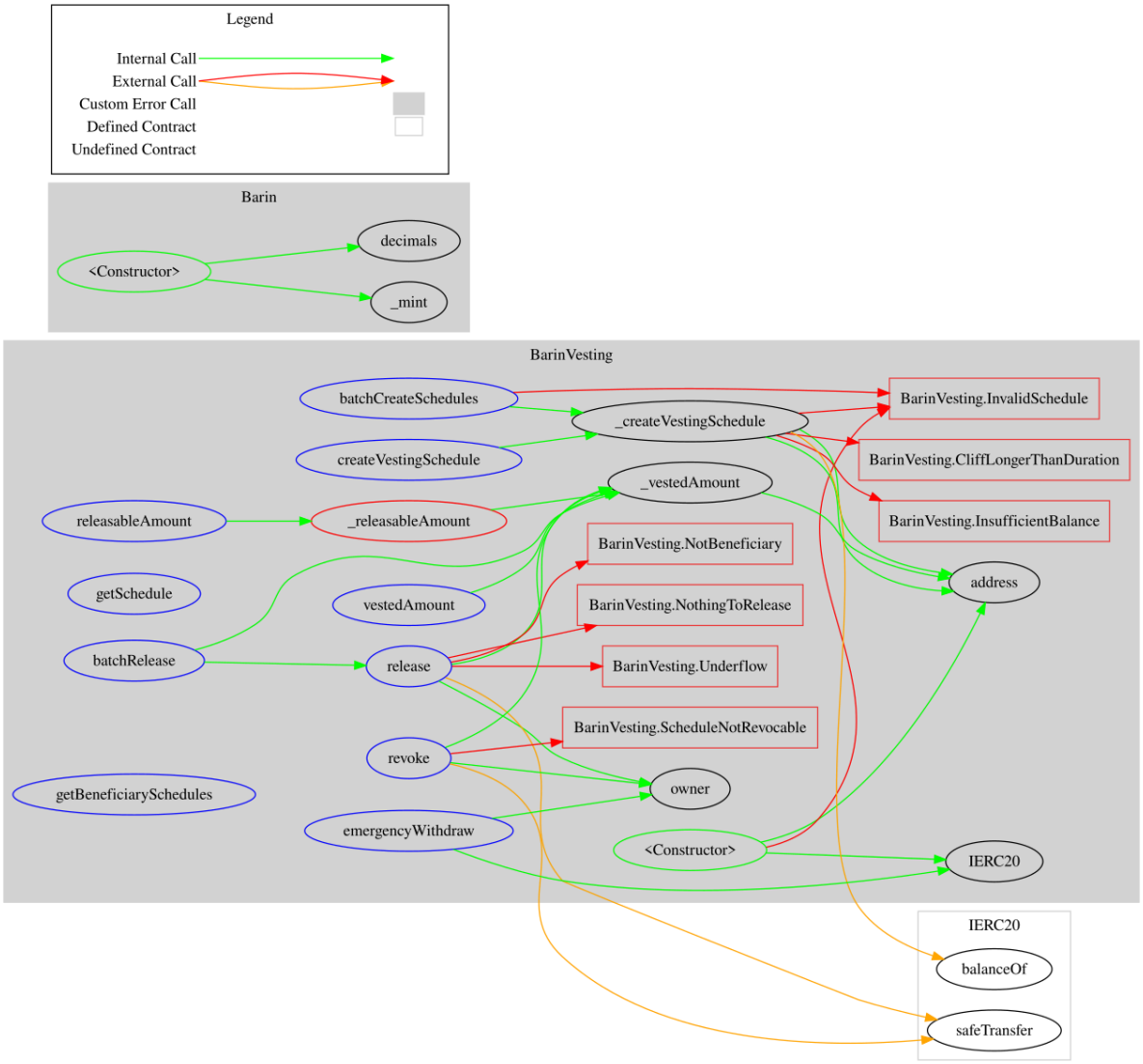
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
BarinVesting	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	createVestingSchedule	External	✓	onlyOwner
	batchCreateSchedules	External	✓	onlyOwner
	_createVestingSchedule	Internal	✓	
	release	External	✓	nonReentrant
	batchRelease	External	✓	-
	revoke	External	✓	onlyOwner
	_releasableAmount	Private		
	_vestedAmount	Private		
	getSchedule	External		-
	releasableAmount	External		-
	vestedAmount	External		-
	getBeneficiarySchedules	External		-
	emergencyWithdraw	External	✓	onlyOwner

Barin	Implementation	ERC20, ERC20Permit		
		Public	✓	ERC20 ERC20Permit

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Barin
<Constructor>
decimals
_mint
BarinVesting
batchCreateSchedules
createVestingSchedule
_createVestingSchedule
BarinVesting.InvalidSchedule
BarinVesting.CliffLongerThanDuration
BarinVesting.InsufficientBalance
_vestedAmount
releasableAmount
_releasableAmount
BarinVesting.NotBeneficiary
address
getSchedule
vestedAmount
BarinVesting.NothingToRelease
batchRelease
release
BarinVesting.Underflow
BarinVesting.ScheduleNotRevocable
revoke
owner
getBeneficiarySchedules
emergencyWithdraw
<Constructor>
IERC20
IERC20
balanceOf
safeTransfer

Summary

Barin Mineral Token contract implements a token and vesting 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

cyberscope.io