Cyberscope

Audit Report

BinoStake

March 2024

binoStakeBNB	8f2bed09053ff034d3a6992f9a8b6d86b443d9cd6159e77b8919f95338f31528
BinoStakeManager	f105a06e5b0798aa59b6308a1773b69008cc2ffbff6541c1f1fe8054b7bbec1a

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	07 Mar 2024

Source Files

Filename	SHA256
binoStakedBNB.sol	8f2bed09053ff034d3a6992f9a8b6d86b44 3d9cd6159e77b8919f95338f31528
BinoStakeManager.sol	f105a06e5b0798aa59b6308a1773b69008 cc2ffbff6541c1f1fe8054b7bbec1a
interfaces/IStakeManager.sol	58c6306a3b5c0b7dcb51eda857db39f2fa 2b335fea4dd7f5dac3b8fba17a59bb
interfaces/INativeStaking.sol	ff40e30f96ede65d4fa42e603974d7b0ee25 5fe6c5bee7221f30106d79a050f9
interfaces/IBinoStakedBNB.sol	61b577739d8148027af465a9e38c1ccc11 d8b6363da244a2a82d2fffc8a35e8e

Overview

Cyberscope audited two contracts of the BinoStake ecosystem, bimoStakedBNB, BinoStakeManager. The first is a common ERC20 token with additional burn and mint functionality. The latter is designed to manage the staking of BNB on the Binance Smart Chain (BSC). It serves as an intermediary layer between users and the native staking contract. The core functionalities of the contract include depositing BNB, delegating funds to the native staking contract, compounding rewards, requesting withdrawals, undelegating funds, and managing the reserve amount for delegation.

Upon initialization, the BinoStakeManager contract allows for the setting of essential parameters such as the BsBnb token address, admin, manager, bot address, reward fee, revenue pool, and validator. Users can deposit BNB into the contract, which in turn mints BsBnb tokens for the user. The bot role is responsible for delegating users' funds to the native staking contract, either with or without reserved BNB.

Additionally, the contract provides functionalities for compound rewards, requesting and claiming withdrawals, undelegating funds, and managing the reserve amount. It also includes features for proposing and accepting a new manager, setting the validator address, adjusting the reward fee, and managing the revenue pool and redirect address.

Overall, the BinoStakeManager contract facilitates the efficient management and interaction of users' staked BNB assets with the native staking contract on the Binance Smart Chain, providing a secure and flexible staking solution.

Upgradability

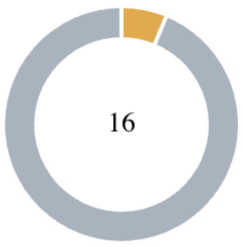
In addition to their core functionalities, both contracts are upgradeable, allowing for the potential modification and enhancement of their implementation and functionality over time. This upgradeability feature enables developers to adapt the contracts to changing requirements, fix bugs, and introduce new features without disrupting its deployed instances or requiring users to migrate to a new contract address.

By leveraging upgradeable contracts, the BinoStake ecosystem can evolve alongside advancements in technology and best practices, ensuring its long-term viability and effectiveness in managing staking activities on the Binance Smart Chain. This flexibility underscores the commitment to maintaining and improving the contracts' capabilities to meet the needs of its users and stakeholders in the dynamic blockchain ecosystem.

Test Deployment

Contract	Explorer
binoStakedBNB	https://testnet.bscscan.com/address/0x621E0Cf03748fEA9D15e263fa4d7cE691c6c5fF6
BinoStakeManager	https://testnet.bscscan.com/address/0x853D1b2f1731f6db649f9dc3Ea763F72a8505750

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	15

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	15	0	0	0

Diagnostics

🟥 Critical 🟧 Medium ● Minor / Informative

Severity	Code	Description	Status
🟧	SUP	Stops Undelegation Process	Unresolved
●	BT	Burns Tokens	Unresolved
●	CR	Code Repetition	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	DPI	Delegation Process Inconsistency	Unresolved
●	MT	Mints Tokens	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MCFD	Missing Claim Function Distinction	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	PMF	Potential Missing Funds	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RVU	Redundant Variable Usage	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	SVII	State Variable Iterable Increase	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L19	Stable Compiler Version	Unresolved

SUP - Stops Undelegation Process

Criticality	Medium
Location	BinoStakeManager.sol#L335,398
Status	Unresolved

Description

The manager role has the authority to menipulate the `reserveAmount` parameter. By manipulating the `reserveAmount` parameter to a substantial value, the `undelegate` method could ask for an amount higher than the current delegated value. This exploitation could disrupt the functionality of the staking platform.

```
IStaking(NATIVE_STAKING).undelegate{value: msg.value}(validator, _amount +
reserveAmount);

function setReserveAmount(uint256 amount) external override onlyManager {
    reserveAmount = amount;
    emit SetReserveAmount(amount);
}
```

Recommendation

The contract could implement a process tha guarantees the correct operation of the undelegate method. The maximum value that should be asked by the undelegate method should be total delegated amount.

BT - Burns Tokens

Criticality	Minor / Informative
Location	binoStakedBNB.sol#L46
Status	Unresolved

Description

The contract stake manager has the authority to burn tokens from a specific address. The stake manager is intented to be the BinoStakeManager contract. However, the contract's admin can modify the stake manager's address to any address. The stake manager may take advantage of it by calling the `burn` function. As a result, the targeted address will lose the corresponding tokens.

We state that `admin` and `stakeManager` privileges are necessary and required for proper protocol operations. Thus, we emphasize the admins to be extra careful with the credentials.

```
function burn(address _account, uint256 _amount)
    external
    override
    onlyStakeManager
{
    _burn(_account, _amount);
}
```

Recommendation

The team should carefully manage the private keys of the stake manager’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.

CR - Code Repetition

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L139,168,194,307
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
uint256 relayFee = IStaking(NATIVE_STAKING).getRelayerFee();
uint256 relayFeeReceived = msg.value;
require(relayFeeReceived == relayFee, "Insufficient RelayFee");
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	binoStakedBNB.sol#L54 BinoStakeManager.sol#L424,439,634
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 setStakeManager(address _address)
    external
    override
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    require(stakeManager != _address, "Old address == new address");
    require(_address != address(0), "zero address provided");

    stakeManager = _address;

    emit SetStakeManager(_address);
}
function setBotRole(address _address) external override {
    require(_address != address(0), "zero address provided");

    grantRole(BOT, _address);
}
...
```

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.

DPI - Delegation Process Inconsistency

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L131,160,299
Status	Unresolved

Description

The contract includes two functions for delegating funds to the native staking contract: `delegate` and `delegateWithReserve` . While the `delegateWithReserve` function incorporates the `reserveAmount` in the delegation process, the delegate function does not. However, there is only one `undelegate` function, which includes the `reserveAmount` . This discrepancy may lead to potential transaction reversion if funds were not delegated with the `delegateWithReserve` function, yet the contract attempts to undelegate them with the undelegate function.

```
function delegate()
    external
    payable
    override
    whenNotPaused
    onlyRole(BOT)
    returns (uint256 _amount)
{}
function delegateWithReserve()
    external
    payable
    override
    whenNotPaused
    onlyRole(BOT)
    returns (uint256 _amount)
{}
function undelegate()
    external
    payable
    override
    whenNotPaused
    onlyRole(BOT)
    returns (uint256 _uuid, uint256 _amount)
{}
```

Recommendation

To ensure consistency and prevent transaction reversion, align the delegation and undelegation processes regarding the `reserveAmount` parameter. Consider modifying the delegate function to either include the `reserveAmount` or create a separate undelegate function that does not require the `reserveAmount` parameter. By implementing these adjustments, the contract can maintain coherence and integrity in both delegation and undelegation operations, enhancing the overall user experience and contract functionality.

MT - Mints Tokens

Criticality	Minor / Informative
Location	binoStakedBNB.sol#L38
Status	Unresolved

Description

The contract stake manager has the authority to mint tokens. The stake manager is intented to be the BinoStakeManager contract. However, the contract's admin can modify the stake manager's address to any address. Additonally, the contract has no cap for the `totalSupply`. The stake manager may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

We state that `admin` and `stakeManager` privileges are necessary and required for proper protocol operations. Thus, we emphasize the admins to be extra careful with the credentials.

```
function mint(address _account, uint256 _amount)
    external
    override
    onlyStakeManager
{
    _mint(_account, _amount);
}
```

Recommendation

The team should carefully manage the private keys of the stake manager’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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L457
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_synFee <= TEN_DECIMALS, "_synFee must not exceed 10000 (100%)");
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MCFD - Missing Claim Function Distinction

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L340,361
Status	Unresolved

Description

Within the contract, the functions `claimUndelegated` and `claimFailedDelegation` both interact with the native staking contract to claim amounts, but their purposes and consequences differ significantly. However, the contract does not provide a clear way to differentiate between these functions, potentially leading to invalid calls and loss of sequence integrity, particularly with the `nextUndelegateUUID` .

```
function claimUndelegated()
    external
    override
    whenNotPaused
    onlyRole(BOT)
    returns (uint256 _uuid, uint256 _amount)
{}
function claimFailedDelegation(bool withReserve)
    external
    override
    whenNotPaused
    onlyRole(BOT)
    returns (uint256 _amount)
{}
```

Recommendation

To mitigate confusion and ensure the correct usage of these functions, the team could implement clearer differentiation mechanisms. The team should consider updating the function names and/or adding additional functionality that clearly indicates their intended purpose. For instance, not allowing the execution of the `claimFailedDelegation` if there is no actual failure during the execution. By enhancing clarity and differentiation, the contract can maintain sequence integrity and minimize the risk of invalid calls or unintended consequences.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L313,394
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 `reserveAmount` must be less than or equal to `totalReserveAmount`.
- The `undegelate` function should be executed only when the `totalBsBnbToBurn` is greater than 0.

```
reserveAmount = amount;
uint256 totalBsBnbToBurn_ = totalBsBnbToBurn; // To avoid Reentrancy attack
_amount = convertBsBnbToBnb(totalBsBnbToBurn_);
```

Recommendation

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

PMF - Potential Missing Funds

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L180,335
Status	Unresolved

Description

The `delegateWithReserve` and `undelegate` functions incorporate a `reserveAmount` parameter intended to facilitate additional BNB delegation or undelegation. However, inconsistencies arise between these functions if the stake manager modifies the `reserveAmount` after a delegation process. This inconsistency results in potential transaction reversion during undelegation, as the increased `reserveAmount` may not be available in the native staking contract.

```
IStaking(NATIVE_STAKING).delegate{value: _amount + msg.value +
reserveAmount}(validator, _amount + reserveAmount);
IStaking(NATIVE_STAKING).undelegate{value: msg.value}(validator, _amount +
reserveAmount);
```

Recommendation

To ensure consistency and prevent transaction reversion, it is essential to synchronize the `reserveAmount` adjustments between the delegation and undelegation processes. The team is advised to revise the code segments and rewrite them, so the `reserveAmount` is consistent throughout the process. By addressing these inconsistencies, the contract can maintain reliability and prevent disruptions to stake delegation and undelegation operations.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L224
Status	Unresolved

Description

The contract sends funds to a `revenuePool` as part of the compound rewards flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the staking manager contract and revert the transaction.

```
AddressUpgradeable.sendValue(payable(revenuePool), fee);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the compound rewards flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RVU - Redundant Variable Usage

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L35,299,340
Status	Unresolved

Description

As part of the undelegation process, the contract incorporates the `reserveAmount` parameter to request an additional BNB amount from the native staking contract, including the user's staked amount. However, in the `claimUndelegated` function, the `reserveAmount` is not utilized, indicating redundancy in its inclusion.

```
uint256 public reserveAmount;
```

Recommendation

To streamline the contract logic and eliminate redundancy, consider removing the `reserveAmount` parameter from the `claimUndelegated` function. Since it is not utilized within this function, its presence serves no functional purpose and may lead to confusion. By removing the redundant parameter, the contract becomes more concise, easier to understand, and less prone to potential errors or misinterpretations.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L174,180
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
_amount + reserveAmount
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

SVII - State Variable Iterable Increase

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L353
Status	Unresolved

Description

Gas optimization refers to the process of reducing the amount of gas required to execute a transaction. Gas is the unit of measurement used to calculate the fees paid to miners for including a transaction in a block on the blockchain.

The `confirmedUndelegatedUUID` variable is incremented within a loop that iterates over undelegated UUIDs. However, this variable could be incremented once after the loop's completion, reducing gas consumption without affecting functionality.

```
for (uint256 i = confirmedUndelegatedUUID; i <= nextUndelegateUUID - 1; i++) {
    BotUndelegateRequest
        storage botUndelegateRequest = uuidToBotUndelegateRequestMap[i];
    botUndelegateRequest.endTime = block.timestamp;
    confirmedUndelegatedUUID++;
}
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L389,394
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
totalReserveAmount += amount
totalReserveAmount -= amount
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BinoStakeManager.sol#L2 binoStakedBNB.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.

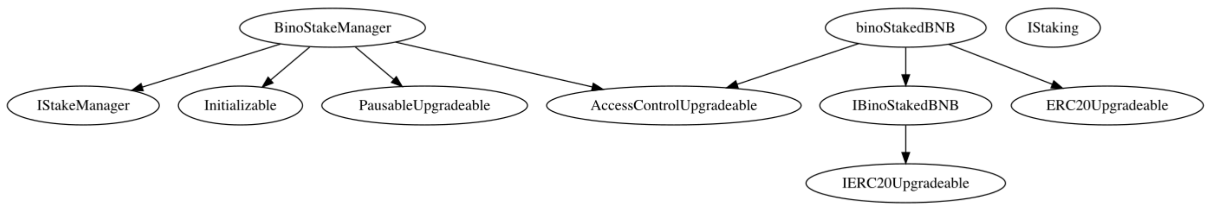
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
binoStakedBNB	Implementation	IBinoStakedBNB, ERC20Upgradeable, AccessControlUpgradeable		
		Public	✓	-
	initialize	External	✓	initializer
	name	Public		-
	symbol	Public		-
	mint	External	✓	onlyStakeManager
	burn	External	✓	onlyStakeManager
	setStakeManager	External	✓	onlyRole
BinoStakeManager	Implementation	IStakeManager, Initializable, PausableUpgradeable, AccessControlUpgradeable		
		Public	✓	-
	initialize	External	✓	initializer
	deposit	External	Payable	whenNotPaused

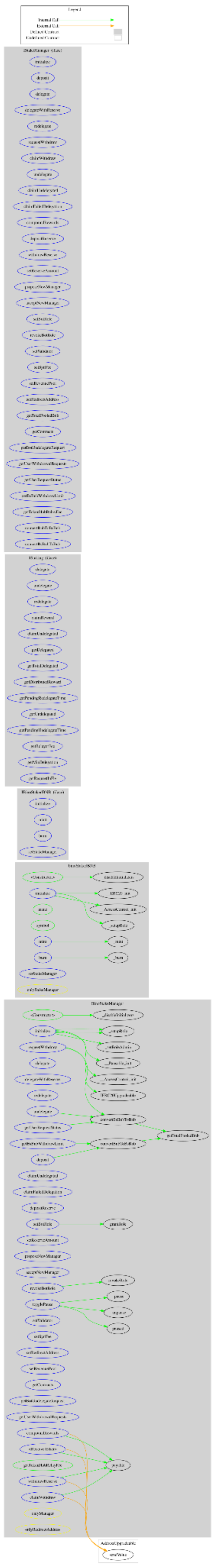
	delegate	External	Payable	whenNotPaused onlyRole
	delegateWithReserve	External	Payable	whenNotPaused onlyRole
	redelegate	External	Payable	whenNotPaused onlyManager
	compoundRewards	External	✓	whenNotPaused onlyRole
	requestWithdraw	External	✓	whenNotPaused
	claimWithdraw	External	✓	whenNotPaused
	undelegate	External	Payable	whenNotPaused onlyRole
	claimUndelegated	External	✓	whenNotPaused onlyRole
	claimFailedDelegation	External	✓	whenNotPaused onlyRole
	depositReserve	External	Payable	whenNotPaused onlyRedirectAddress
	withdrawReserve	External	✓	whenNotPaused onlyRedirectAddress
	setReserveAmount	External	✓	onlyManager
	proposeNewManager	External	✓	onlyManager
	acceptNewManager	External	✓	-
	setBotRole	External	✓	-
	revokeBotRole	External	✓	-
	setValidator	External	✓	onlyManager
	setSynFee	External	✓	onlyRole
	setRedirectAddress	External	✓	onlyRole

	setRevenuePool	External	✓	onlyRole
	getTotalPooledBnb	Public		-
	getContracts	External		-
	getBotUndelegateRequest	External		-
	getUserWithdrawalRequests	External		-
	getUserRequestStatus	External		-
	getBsBnbWithdrawLimit	External		-
	getTokenHubRelayFee	Public		-
	convertBnbToBsBnb	Public		-
	convertBsBnbToBnb	Public		-
	togglePause	External	✓	onlyRole
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

BinoStake contract implements a token and staking mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

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