BEOSIN
Web3 Security & Compliance

Tizi

Smart Contract Security Audit

No. 202601291648

Jan 29th, 2026

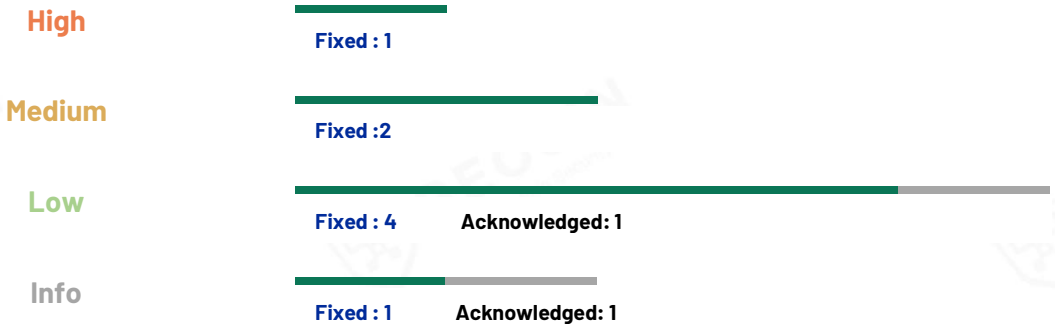
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 High, 2 Medium, 5 Low and 2 Info risks were identified in the Tizi project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



- **Project Description:**

Tizi is an innovative cross-chain DeFi protocol designed to deliver efficient stablecoin management and sustainable yield generation. At its core lies Tizi Dollar (TD), a USDC-pegged stablecoin that users can mint by depositing USDC through the DepositHelper contract on the Base chain. Deposited funds are exchanged for TD at a 1:1 ratio (adjusted for a small fee), and users may subsequently stake TD to receive Staked TD (stTD) in order to earn protocol-generated yields.

The protocol's architecture includes several key components:

DepositHelper manages user deposits and withdrawals. When sufficient liquidity exists in the MainVault, USDC is transferred directly to the user upon TD redemption. In cases of insufficient liquidity, an NFT is minted to record the pending withdrawal amount. Once administrators transfer funds to the NFTVault, users can redeem the corresponding USDC. This mechanism ensures robust liquidity control while maintaining user redemption rights.

StrategyManager (on Base) and SubStrategyManager (on other chains) govern the deployment and activation of yield-generating strategies across multiple blockchains. Strategies undergo a mandatory cooldown period (default: 3 days) before activation, and activation is conditional on verified available liquidity, coordinated via LayerZero cross-chain messaging.

TiziDollar (TD) implements a rebase mechanism to reflect real-time changes in protocol net asset value. Positive yield results in gradual minting and linear distribution to stTD holders; negative growth is first absorbed by an insurance pool to protect principal holders.

Staked TD (stTD) follows the ERC-4626 standard and supports both mainnet and child-chain deployments. It features a configurable unstaking cooldown (default: 7 days) and linear release of accrued yield (default: 7 days), with 10% of protocol profits allocated to a designated recipient.

StakeVault securely holds TD in a pending state during staking operations.

1 Overview

1.1 Project Overview

Project Name	Tizi
Project Language	Solidity
Platform	Base
Contract Address	https://github.com/tizimoney/Tizi-contract
Commit hash	920bb66a1f38a1a59d89d54688eaef894f2cdaaa (Origin) 7c720ab3a2f6eeb025a2003857d4f991651cbd44 (Latest)

1.2 Audit Overview

Audit work duration: Jan 5 - Jan 29, 2026

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a function of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Tizi-01	Yield Loss Handling Flaws	High	Fixed
Tizi-02	Unstake Queue Overflow Without Error	Medium	Fixed
Tizi-03	Centralized Risk	Medium	Fixed
Tizi-04	Insufficient Validations	Low	Fixed
Tizi-05	Linear Address Check May Cause Gas Exhaustion	Low	Acknowledged
Tizi-06	Strategy Can Be Activated Repeatedly	Low	Fixed
Tizi-07	Signature Replay Vulnerability	Low	Fixed
Tizi-08	Negative Strategy Value Not Handled Correctly in Total Calculation	Low	Fixed
Tizi-09	Stuck ETH in TimelockController	Info	Acknowledged
Tizi-10	Redundant Code	Info	Fixed

Finding Details:

[Tizi-01] Yield Loss Handling Flaws

Severity Level	High
Type	Business Security
Lines	ChildstTD.sol#L225-235
Description	1.In the `addYield` function of the ChildstTD contract, two issues arise when processing losses that require partial consumption: 2.When `negativeGrowth` is true and the process enters step 2, `releaseAmount` is updated, but `lastYieldTime` is not, which may disrupt the timing of subsequent yield releases. If any `remainingAmount` remains after step 2, it is silently discarded without being carried forward to future operations, potentially affecting the integrity of the economic model.
Recommendation	It is recommended to update `lastYieldTime` when modifying `releaseAmount` in the negative growth path, and to implement appropriate handling for any remaining amount according to the intended protocol design.
Status	**Fixed.** Point 1: Fixed — `lastYieldTime` is now updated in step 1. Point 2: Per project team: negative growth is covered in sequence from (1)

```
            // step2: check unreleased amount
            if(remainingAmount > 0) {
                uint256 unreleased = getUnreleasedAmount();
                uint256 toBurnFromUnreleased = Math.min(unreleased,
remainingAmount);
                if(toBurnFromUnreleased > 0) {
                    releaseAmount = releaseAmount -
toBurnFromUnreleased;
                    stakedTDAmount = stakedTDAmount -
toBurnFromUnreleased;
                    remainingAmount = remainingAmount -
toBurnFromUnreleased;
                    ITD(asset()).burnForYield(toBurnFromUnreleased);
                }
            }
```

insurance pool → (2) unreleased amount → (3) released staking pool. If still insufficient, no further action is taken.

```
            // step1: check unreleased amount
            if(remainingAmount > 0) {
......
                if(toBurnFromUnreleased > 0) {
                    _updateNegativeReleaseAmount(toBurnFromUnreleased
);
......
                }
            }
```

[Tizi-02] Unstake Queue Overflow Without Error

Severity Level	**Medium**
Type	Business Security
Lines	ChildstTD.sol#L330-336
Description	In the `unstake` function of the ChildstTD contract, when a user's unstake queue already contains 7 entries, a new unstake request will fail to be added to the queue but still updates `totalShares` without reverting the transaction. `for(uint256 i = 0; i < userUnstakingInfo.length; ++i) { if(userUnstakingInfo[i].amount == 0 && userUnstakingInfo[i].endTime == 0) { userUnstakingInfo[i].endTime = block.timestamp + unstakingPeriod; userUnstakingInfo[i].amount += assets; break; }`
Recommendation	It is recommended to check the loop index after attempting to add the unstake record; if the index exceeds the queue length (indicating failure to add), revert the transaction with an appropriate error message.
Status	**Fixed.** Added `found` check to detect limit exceedance and return appropriate error message `bool found = false; for(uint256 i = 0; i < userUnstakingInfo.length; ++i) { if(userUnstakingInfo[i].amount == 0 && userUnstakingInfo[i].endTime == 0) { userUnstakingInfo[i].endTime = block.timestamp + unstakingPeriod; userUnstakingInfo[i].amount += assets; found = true; break; } } if (!found) { revert MaxUnstakeQueue(msg.sender); }`

[Tizi-03] Centralized Risk

Severity Level	**Medium**
Type	Business Security
Lines	All main contract
Description	Administrators in the project possess extremely high privileges across nearly all critical operations. In the StrategyManager contract, the admin can arbitrarily add and activate strategies. In TiziDollar, the admin can influence the minting process by modifying the helper address. Additionally, LayerZero cross-chain configuration parameters in almost all major contracts are controlled and set exclusively by the admin. Should the admin private key be compromised, malicious configuration changes could result in severe asset loss.
Recommendation	It is recommended to implement multi-signature wallets (multisig) or similar decentralized mechanisms to manage critical administrative privileges throughout the project.
Status	**Fixed.** The project team has stated that multi-signature (multi-sig) control will be implemented for the admin privileges.

[Tizi-04] Insufficient Validations

Severity Level	Low
Type	Business Security
Lines	MainVault.sol#L247-255
Description	Several administrator-controlled configuration functions lack sufficient input validation: 1. In the Vault contract does not prevent duplicate entries.

```
    function addAllowedVaults(address[] memory vaults) external
onlyAdmin {
        (, bool isAllow) =
IDepositHelper(depositHelper).calculateLiquidity();
        require(isAllow, "No liquidity to add new SubVault!");
        require(vaults.length > 0, "Please input at least one address");
        for(uint i = 0; i < vaults.length; ++i) {
            require(vaults[i] != address(0), "Wrong vault address");
            allowedVaults[vaults[i]] = true;
        }
    }
```

2. In the TimeLock contract, `_minDelay` can be set to 0.

```
    function updateDelay(uint256 newDelay) external virtual {
        address sender = _msgSender();
        if (sender != address(this)) {
            revert TimelockUnauthorizedCaller(sender);
        }
        emit MinDelayChange(_minDelay, newDelay);
        _minDelay = newDelay;
    }
```

3. The `profitNumerator` parameter has no upper bound and may be configured to exceed 100%.

```
      function setProfitNumerator(uint256 newProfitNumerator)
external onlyAdmin {
        if (block.chainid != mainChainId) {
            revert UnsupportedChain(block.chainid);
        }
        require(newProfitNumerator != profitNumerator, "Wrong
profitNumerator");
```

```
        profitNumerator = newProfitNumerator;
        emit SetNewProfitNumerator(newProfitNumerator);
```

Recommendation It is recommended to implement proper validation checks in these setter functions.

Status **Fixed.** 1. No security impact.

2. Added a minimum value of 1 day.

```
         require(newDelay >= IStakedTD(stakedTD).unstakingPeriod() + 1
days,
            "Daley should larger than unstake period");
```

3. Added an upper limit of 100.

```
        require(newProfitNumerator != profitNumerator &&
newProfitNumerator <= 100,
            "Wrong profitNumerator");
```

[Tizi-05] Linear Address Check May Cause Gas Exhaustion

Severity Level	Low
Type	Business Security
Lines	MainTokenStats.sol#L211-222
Description	In the Statistics contract, checking whether an address has already been added is done via array iteration. As the array grows, this may lead to excessive gas consumption and could cause the update function to fail.
Recommendation	It is recommended to replace the array iteration with a mapping-based approach to efficiently track and check whether an address has been added.
Status	**Acknowledged.**

```
    function isContractAddressAdded(
      uint256 chainID,
      address contractAddress
   ) internal view returns (bool) {
      address[] memory existingAddresses =
chainStrategyKeys[chainID];
      for (uint256 i = 0; i < existingAddresses.length; i++) {
          if (existingAddresses[i] == contractAddress) {
              return true;
          }
      }
      return false;
   }
```

[Tizi-06] Strategy Can Be Activated Repeatedly

Severity Level	Low
Type	Business Security
Lines	StrategyManager.sol#254-268
Description	In both StrategyManager and its corresponding SubStrategyManager, the activation logic only verifies whether the strategy exists and whether the cooldown period has elapsed. It does not check whether the strategy is already active. Repeated activation of the same strategy will cause duplicate entries in the `activeStrategyAddresses` array, potentially polluting the data structure and leading to incorrect behavior or errors in subsequent query or iteration functions.
Recommendation	It is recommended to add an explicit check to ensure the strategy is not already active before performing the activation.
Status	**Fixed.** Added check for whether the added strategy is activated.

```
    function activateStrategy(
        uint256 chainID,
        address strategyAddress
    ) external onlyAdmin {
        require(
            _isContract(strategyAddress) == true,
            "The address must be a contract"
        );
        require(
            strategies[chainID][strategyAddress].exists,
            "Strategy does not exist"
        );
        require(
            block.timestamp -
strategies[chainID][strategyAddress].addedTime >= cooldownTime,
            "Adding time is less than the cooldown time."
        );
```

```
        require(
           strategies[chainID][strategyAddress].active == false,
            "The strategy has been activated."
        );
```

[Tizi-07] Signature Replay Vulnerability

Severity Level	Low
Type	Business Security
Lines	SubStrategyManager.sol#L316-341
Description	In the `_lzReceive` function of SubStrategyManager (and similarly in the main chain's LayerZero-related contracts), signature verification only checks whether the recovered signer is an admin address. It does not verify nonce, source chain ID, destination chain ID, or the originating contract address. This design allows valid signatures from the main chain to be replayed or reused on child chains, or even across different child chains.

```
 function _lzReceive(
        Origin calldata,
        bytes32,
        bytes calldata payload,
        address,  // Executor address as specified by the OApp.
        bytes calldata  // Any extra data or options to trigger on
receipt.
    ) internal override {
        (bytes memory signedMessage, bytes memory message) = abi.decode(
            payload,
            (bytes, bytes)
        );
        bytes32 hashMessage = keccak256(message);
        bytes32 ethMessage = toEthSignedMessageHash(hashMessage);
        address signer = ethMessage.recover(signedMessage);
        require(
            _authorityControl.hasRole(
                _authorityControl.DEFAULT_ADMIN_ROLE(),
                signer
            ),
            "Not authorized"
        );
        (bool canActivate, uint256 time) = abi.decode(message, (bool,
uint256));
        liquidityInfo.canActivate = canActivate;
        liquidityInfo.time = time;
        emit SignedMessageVerified(signer, hashMessage);
```

Recommendation	It is recommended to Include relevant identifiers (nonce, source chain ID, destination chain ID, sender contract address, etc.) in the signed message payload, and perform strict validation of these fields inside the `_lzReceive` function to prevent replay and cross-chain misuse.
Status	**Fixed.** Added the target address, a random nonce, and a deadline to the message, along with the corresponding validation checks.

```
        (
            address targetContract,
            uint256 nonce,
            uint256 deadline,
            bytes memory code
        ) = abi.decode(
            message,
            (address, uint256, uint256, bytes)
        );

        require(targetContract == address(this), "Wrong target");
        require(block.timestamp <= deadline, "Signature expired");
        require(!usedNonce[nonce], "Nonce used");
        usedNonce[nonce] = true;
```

[Tizi-08] Negative Strategy Value Not Handled Correctly in Total Calculation

Severity Level	Low
Type	Business Security
Lines	MainTokenStats.sol#L323-337
Description	Although `totalStrategyValue` is declared as an int type to accommodate profit and loss, the final accumulation into `chainTotalValues` (a uint) is performed using addition after casting to uint. When `totalStrategyValue` is negative, this results in incorrect arithmetic behavior (underflow or wrapping), leading to inaccurate total asset value calculations.
Recommendation	At the end of the loop, it is recommended to explicitly check the sign of `totalStrategyValue` and perform addition or subtraction accordingly to ensure correct statistical aggregation.
Status	**Fixed.** Changed the type to int256 and added validation checks for positive and negative values.

```
    for (uint256 k = 0; k < tokenInfos.length; k++) {
                    uint256 tokenAmount = tokenInfos[k].tokenAmount;
                    uint256 tokenValue = tokenInfos[k].tokenValue;
                    if(tokenInfos[k].negativeGrowth) {
                        totalStrategyValue -= int256(tokenAmount *
tokenValue);
                    } else {
                        totalStrategyValue += int256(tokenAmount *
tokenValue);
                    }
                }
                strategyTotalValues[chainID][strategyAddress] =
                    uint256(totalStrategyValue) /
                    10 ** 18;
                totalChainValue += uint256(totalStrategyValue) / 10 **
18;
```

```
                if(totalStrategyValue > 0) {
                    strategyTotalValues[chainID][strategyAddress] =
                    totalStrategyValue /
                    10 ** 18;
                    totalChainValue += uint256(totalStrategyValue) / 10
```

```
** 18;
                } else {
                    strategyTotalValues[chainID][strategyAddress] =
                    totalStrategyValue /
                    10 ** 18;
                    totalChainValue -= uint256(-totalStrategyValue) /
10 ** 18;
                }
```

[Tizi-09] Stuck ETH in TimelockController

Severity Level	Info
Type	Business Security
Lines	TimelockController.sol#L413-415

Description	The TimelockController contract does not implement any function to withdraw or recover native ETH that may accumulate in the contract. If, during the execution of an operation, the target contract returns excess ETH (e.g., due to refund logic), these funds will become permanently locked inside the Timelock contract.
	`function _execute(address target, uint256 value, bytes calldata data) internal virtual {`
	`(bool success, bytes memory returndata) = target.call{value: value}(data);`
	`Address.verifyCallResult(success, returndata);`
	`}`

Recommendation	It is recommended to add a secure function (preferably restricted to a trusted role or via timelock) that allows withdrawal of native ETH from the contract.
Status	**Acknowledged.**

[Tizi-10] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	MainVault.sol#L24-34
Description	The `depositForBurnWithHook` interface function referenced in the Vault contracts (including MainVault and SubVault) is not used anywhere in the current codebase, resulting in unnecessary interface bloat.

```
    function depositForBurnWithHook(
        uint256 amount,
        uint32 destinationDomain,
        bytes32 mintRecipient,
        address burnToken,
        bytes32 destinationCaller,
        uint256 maxFee,
        uint32 minFinalityThreshold,
        bytes calldata hookData
    ) external;
}
```

Recommendation	It is recommended to remove the unused `depositForBurnWithHook` function from the relevant interfaces to reduce code complexity and improve maintainability.
Status	**Fixed.** Redundant code has been removed.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Critical**

Critical impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is a leading blockchain security and compliance technology company established in 2018. Being focused on blockchain ecosystem security and compliance, it has developed a product matrix including Beosin KYT, Beosin Trace, and Stablecoin Monitor, which have obtained international certifications such as ISO 27001 and SOC 2. Beosin's core products have been applied for over 70 intellectual property rights, and the company has participated in the development of multiple international standards related to blockchain security. It was among the first batch of enterprises selected for the Cyberport Incubation Programme. Its business covers professional code security audit services for blockchain ecosystems, anti-money laundering compliance technology services for exchanges, financial institutions, and payment institutions, and virtual asset tracing and investigation services for law enforcement and regulatory authorities.

As one of the earliest companies to apply formal verification to blockchain security, Beosin offers professional blockchain and smart contract security audit services. Beosin has audited over 4,500 smart contracts and blockchain projects and has become the official security partner for several renowned blockchains, including BNB Chain, TON, Soneium, Manta Network, Sonic SVM, and SOON Network.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin