BEOSIN

Blockchain Security

# Aeth Staking

Smart Contract Security Audit

V1.0

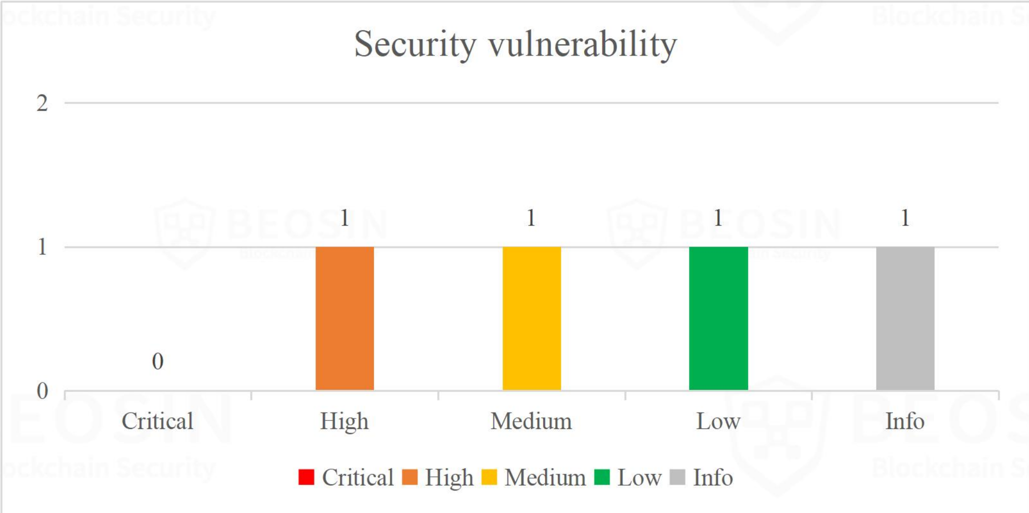
No. 202211151729

Nov 15th, 2022

# Contents

# Summary of Audit Results

**After auditing, 1 High-risk items and 1 Medium and 1 Low-risk items and 1 Info items were identified in the Aeth Staking project.** Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:




*Notes:

- **Risk Description:**

1. After auditing the Aeth Staking project, the project team confirmed several file will not be used. ANRK.sol, depositContract.sol is only used for testing, Governance_R3.sol, AETHF.sol and the library unisawpinteractLib.sol have been removed or have been deprecated, AnkrDeposit_R3.sol may not be used (include all older versions).

2. For cross-chain bridge address setting, cross-chain address has the privileges to mint and destroy the AETH pegged token, it may cause loss if the cross-chain bridge is not secure or the validation node is under attack. The project team has no plan to change the cross-chain address and already removed related function. The cross-chain bridge address will no longer to change in this circumstance.

3. For the user, the funds provided by the provider will be locked in the stake contract, but the operator can force the Provider to exit,and reduce the rewards as punishment. Provider can withdraw the pegged token only after forced out, share is excluded. In stake contracts, Assets frozen by *topUpANKR* function can only be unfrozen by forceAdminProviderExit through Operator (the ANKR.sol associated with this in the project will not be used), and this function only can be called after setting the staking contract address, which is designed and expected by project team.

- **Project Description:**

**1. Basic Token Information**

AETH token (from AETH_R16.sol, token name will be determined after deployment) is mintable and burnable, Decimals are 18. FETH token(from FETH_R16.sol)Depends on ratio in AETH_R16 contract, cannot mintable and burnable, AETH token and FETH token with no pre-minted tokens.

**2. Business overview**

Aeth Staking is a multi-strategy staking project that uses platform tokens for staking in related deployed chain. Regular stake will receive AETH pegged token, while provider strategy stake will receive rewards (with a minimum stake limit). This project also support cross-chain stake. The frozen ANKR token assets will be used as insurance assets for provider. All platform token assets stake to the GlobalPool contract can be transferred to the beacon contract as staking by Operator after reaching 32ether.

For the privileges of Operator and Owner, Operator can force exit provider by *forceAdminProviderExit* function, and reset the provider's ledger by *resetLockedEthForProviders* function. Owner can update AETH_R16 contract address, FETH_R16 contract address, as well as AnkrDeposit contract address. Operator can change any value through changeConfiguration in Governance_R3.sol, which has centralization risk. The project team mentioned that they will not use Governance_R3 contract.

The minimum stake limitation in the staking pool can only be set through Governance contract after initialization. Governance contract post proposal and voting for making decision. The votes of proposal are determined by funds ratio, and all currently available funds will be locked in the voting process.

For AETH token, it can only be minted and destroyed by the staking pool contract and cross-chain bridge contract. Owner and operator have the privileges to set ratio, which is reserved to prevent bad rates by the operator's back-end. FETH is futures contract that will based on the AETH contract with the ratio rate., no fee will be charged if the rewards is locked through the staking pool in FETH contract, but fees will be charged if use the *lockShares* function directly. Owner can change the fees ratio by setting *_swapFeeRatio* function , which should not exceed 1%.

# 1 Overview

## 1.1 Project Overview

| **Project Name** | Aeth Staking |
|---|---|
| **File Hash(SHA-256)** | AETH_R16.sol<br>063ffc976977c874e85c7801650c46384242355ef1fd2be7cd16a7955852e849(Initial)<br>f0042546faebd876921ef9fa4ac6cdeffc829eb41ccf5186f72b3666f5996bcd(Latest)<br>FETH_R16.sol<br>db53353bdecd14ff32264d350929f86fe01b16eeefba45e41c532804e334c3ef(Initial)<br>b8f4cb2aebc56cc5f3a51643c7edd23032c19d26c48f39c728ce9b9a7f2ebae5(Latest)<br>GlobalPool_R39.sol<br>7a5528a1a56b262af0fed985874148e6b9777375b2e95c481accd94cd53c6613(Initial)<br>1daceb9e958dbb1a645d4493b42f29fcd0117874bd37baef534e191e7f693d41(Latest)<br>Config.sol<br>19af36d9d251e32f7301f24bddd67eaeb614c1419ea879eb18399dae34852717(Initial)<br>19af36d9d251e32f7301f24bddd67eaeb614c1419ea879eb18399dae34852717(Latest)<br>AnkrDeposit.sol<br>05d35904453bd26e1adeba30710729326c2ec058b2b2b32ff29788c299802077(Initial)<br>2b84d9d3ceadcf4c21187602dbe3ddbfad472ea56136a2368f885e228f350b83(Latest) |

## 1.2 Audit Overview

Audit work duration：Nov 9, 2022 – Nov 15, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

# 2 Findings

| Index | Risk description | Severity level | Status |
|---|---|---|---|
| Aeth Staking-1 | *setOwnership* Function can be called arbitrarily | High | Fixed |
| Aeth Staking-2 | Centralization risk | Medium | Fixed |
| Aeth Staking-3 | Operator or Owner privileges too high | Low | Partially Fixed |
| Aeth Staking-4 | Redundant code | Info | Partially Fixed |

**Status Notes:**

- Aeth Staking-3 is partially fixed and unfixed part will affect the calculation of FETH contract tokens after staking.
- Aeth Staking-4 is partially fixed and this is to support TransparentProxy's upgrade strategy.

# Finding Details:

## [Aeth Staking-1] *setOwnership* function can be called arbitrarily

**Severity Level**: High

**Type**: Business Security

**Lines**: Ownable.sol #L79-82

**Description**: The library file ./lib/Ownable.sol used in FETH_R16 and after calling the *renounceOwnership* function to renounce the owner, any user can call *setOwnership* to become the new Owner.

```
79    function setOwnership() external {
80        require(_owner == address(0));
81        _owner = msg.sender;
82    }
```

Figure 1 The source code of *setOwnership* function

**Recommendations**: It is recommended to delete.

**Status**: Fixed.

## [Aeth Staking-2] Centralization risk

| | |
|---|---|
| **Severity Level** | Medium |
| **Type** | Business Security |
| **Lines** | AETH_R16.sol #L72-86, GlobalPool_R39.sol #L378-382 |
| **Description** | Owner can mint or burn coins arbitrarily in the *burn*, *mint*, and *mintApprovedTo* function in AETH contract. Owner also can arbitrarily set the address of the cross-chain bridge in *changeCrossChainBrigde* function, which may lead to incorrect fund records or calls failure of cross-chain related function. Furthermore, Owner or operator can set the permission of any address to addressAllowed modifier by *allowAddressForFunction* function. When setting the address, the *freeze* function or the *unfreeze* function can be called to freeze or unfreeze the assets of any address. |

```
72    function burn(address account, uint256 amount) external {
73        require(msg.sender == address(_bscBridgeContract) || msg.sender == owner() || msg.sender == address(_globalPoolContract),
74        _burn(account, amount);
75    }
76
77    function mint(address account, uint256 amount) external returns (uint256 _amount) {
78        require(msg.sender == address(_bscBridgeContract) || msg.sender == owner() || msg.sender == address(_globalPoolContract),
79        _mint(account, amount);
80    }
81
82    function mintApprovedTo(address account, address spender, uint256 amount) external {
83        require(msg.sender == address(_bscBridgeContract) || msg.sender == owner() || msg.sender == address(_globalPoolContract),
84        _mint(account, amount);
85        _approve(account, spender, amount);
86    }
```

Figure 2 The source code of *burn, mint, mintApprovedTo* functions(not fixed)

```
378    function changeCrossChainBridge(address crossChainBridgeAddress) public onlyOwner
379        address prevValue = _crossChainBridge;
380        _crossChainBridge = crossChainBridgeAddress;
381        emit CrossChainBridgeChanged(prevValue, crossChainBridgeAddress);
382    }
```

Figure 3 The source code of *changeCrossChainBrigde* function

| | |
|---|---|
| **Recommendations** | It is recommended to delete. |
| **Status** | Fixed. The *changCrossChain* function has been removed. |

```
72    function burn(address account, uint256 amount) external {
73        require(msg.sender == address(_bscBridgeContract) || msg.sender == address(_globalPoolContract), 'Not allowed');
74        _burn(account, amount);
75    }
76
77    function mint(address account, uint256 amount) external returns (uint256 _amount) {
78        require(msg.sender == address(_bscBridgeContract) || msg.sender == address(_globalPoolContract), 'Not allowed');
79        _mint(account, amount);
80    }
81
82    function mintApprovedTo(address account, address spender, uint256 amount) external {
83        require(msg.sender == address(_bscBridgeContract) || msg.sender == address(_globalPoolContract), 'Not allowed');
84        _mint(account, amount);
85        _approve(account, spender, amount);
86    }
```

Figure 4 The source code of *burn, mint, mintApprovedTo* functions(fixed)

## [Aeth Staking-3] Operator or Owner privileges too high

**Severity Level** Low

**Type** Business Security

**Lines** AETH_R16.sol #L49-60, GlobalPool_R39.sol #L342-364

**Description** The owner has no restrictions on setting the rate of the ratio. Although operators limit to update ratio for a single operation, there is no limit on the actual range of values on multi-operation in AETH_R16 contract. Meanwhile, after updating the address of the AETH pegged tokens contract and the address of the stake contract, some users' funds cannot be withdrawn directly.

```
49      function updateRatio(uint256 newRatio) public onlyOperator {
50          // 0.001 * ratio
51          uint256 threshold = _ratio.div(1000);
52          require(newRatio < _ratio.add(threshold) || newRatio > _ratio.sub(threshold), "New ratio should be in limits");
53          _ratio = newRatio;
54          emit RatioUpdate(_ratio);
55      }
56
57      function repairRatio(uint256 newRatio) public onlyOwner {
58           ratio = newRatio;
59          emit RatioUpdate(_ratio);
60      }
```

Figure 5 The source code of *updateRatio, repairRatio* functions

```
342     function updateAETHContract(address payable aEthContract) external onlyOwner {
343         address prevValue = address(_aethContract);
344         _aethContract = IAETH(aEthContract);
345         emit AETHContractChanged(prevValue, aEthContract);
346     }
347
348     function updateFETHContract(address payable fEthContract) external onlyOwner {
349         address prevValue = address(_fethContract);
350         _fethContract = IFETH(fEthContract);
351         emit FETHContractChanged(prevValue, fEthContract);
352     }
353
354     function updateConfigContract(address configContract) external onlyOwner {
355         address prevValue = address(_configContract);
356         _configContract = IConfig(configContract);
357         emit ConfigContractChanged(prevValue, configContract);
358     }
359
360     function updateStakingContract(address stakingContract) external onlyOwner {
361         address prevValue = address(_stakingContract);
362         _stakingContract = IStaking(stakingContract);
363         emit StakingContractChanged(prevValue, stakingContract);
364     }
```

Figure 6 The source code of *updateAETHContract, updateFETHContract, updateConfigContract, updateStakingContract* functions

**Recommendations** It is recommended to use multi-signature wallets or DAO governance.

**Status** Acknowledged.

## [Aeth Staking-4] Redundant code

| | |
|---|---|
| **Severity Level** | Info |
| **Type** | Coding Conventions |
| **Lines** | AETH.sol #L67-69, GlobalPool_R39.sol #L250-252, L282-284, L301-312, L63-67, L74-77, L204-210 AnkrDeposit_R3.sol #L223, |
| **Description** | Variables not used in the function or redundant condition code. |

```
    function isRebasing() external pure returns (bool) {
        return false;
    }
```

Figure 7 The source code of *isRebasing* function(remain)

```
    function slashingsOf(address provider) public view returns (uint256) {
        return 0;
    }
```

Figure 8 The source code of *slashingsOf* function(remain)

```
function _addNewLockToUser(address user, uint256
    uint256 deposits = depositsOf(user);
    uint256 lockedDeposits = lockedDepositsOf(use
    if (amount <= lockedDeposits) {              //
        return;
    }
    amount = amount.sub(lockedDeposits);
    require(amount <= deposits, "Ankr Deposit#_ad

    require(getConfig(_lockEndsAt_, lockId) == 0,
    if (amount == 0) return;
    // set ends at property for lock
    setConfig(_lockEndsAt_, lockId, endsAt);
    // set amount property for lock
    setConfig(_lockAmount_, lockId, amount);
    setConfig(_lockTotal_, user, getConfig(_lockT

    // set lock id
    _userLocks[user].push(lockId);
}
```

Figure 9 The source code of *_addNewLockToUser* function(not fixed)

```
    /*function providerExit() public whenNotPaused("providerExit") {
        _providerExitFor(msg.sender);
    }*/

    function _forceProviderExitFor(address provider) internal {
        uint256 claimableRewards = _aETHProviderRewards[provider];
        if (claimableRewards > 0) {
            _aETHProviderRewards[provider] = _aETHProviderRewards[provider].sub(claimableRewards);
            _aETHRewards[provider] = _aETHRewards[provider].add(claimableRewards);
        }
        uint256 frozenDeposits = _stakingContract.frozenDepositsOf(provider);
        uint256 lockedDeposits = _stakingContract.lockedDepositsOf(provider);
        uint256 ankrDepositBalance = frozenDeposits.sub(lockedDeposits);
        if (ankrDepositBalance > 0) {
            require(_stakingContract.unfreeze(provider, ankrDepositBalance), "Failed to unfreeze");
        }
        emit ProviderExited(provider);
    }

    function _providerExitFor(address provider) internal {
        // check that provider has enough rewards for exit
        uint256 claimableRewards = _aETHProviderRewards[provider];
        require(claimableRewards > 0, "Provider balance should be positive for exit");
        // remember recent exit block (there is a soft lock for exit)
        _exits[provider] = block.number;
        // make the funds claimable
        _aETHProviderRewards[provider] = _aETHProviderRewards[provider].sub(claimableRewards);
        _aETHRewards[provider] = _aETHRewards[provider].add(claimableRewards);
        // emit event
        emit ProviderExited(provider);
    }
```

Figure 10 The source code of *_providerExitFor* function

```
    uint256 oldReward = _rewards[staker].sub(_claims[staker]);
    if (oldReward > 0) {
        _claims[staker] = _claims[staker].add(oldReward);
    }
```

Figure 11 The source code of variables not used(remain)

```
    // deleted fields
    address[] private _pendingStakers; // deleted
    uint256 private _pendingAmount; // deleted
    uint256 private _totalStakes; // deleted
    uint256 private _totalRewards; // deleted

    IAETH private _aethContract;
    IStaking private _stakingContract;
    SystemParameters private _systemParameters;
    address private _depositContract;

    // deleted fields
    address[] private _pendingTemp; // deleted
    uint256[50] private __gap; // deleted
    uint256 private _lastPendingStakerPointer; // deleted

    IConfig private _configContract;

    // deleted fields
    mapping(address => uint256) private _pendingEtherBalances; // delet

    address private _operator; // slot:322

    // deleted fields
    mapping(address => uint256[2]) private _fETHRewards; // deleted

    mapping(address => uint256) private _aETHRewards;
    IFETH private _fethContract;

    // deleted fields
    uint256 private _fethMintBase; // deleted

```

Figure 12 The source code of variables not used(remain)

```
    function claimableFETHRewardOf(address staker) public view returns (uint25
        return claimableAETHRewardOf(staker).mul(1e18).div(_aethContract.ratio
    }

    function claimableAETHFRewardOf(address staker) public view returns (uint2
        return claimableFETHRewardOf(staker);
    }
```

Figure 13 The source code of *claimableFETHRewardOf, claimableAETHFRewardOf* functions

| | |
|---|---|
| **Recommendations** | It is recommended to delete. |
| **Status** | Fixed. Some redundant variables or unused functions are not removed to preserve the layout in TransparentProxy. |

```
function _addNewLockToUser(address user, uint256 amount, uint256 endsAt, uint256 lockId) internal {
    uint256 deposits = depositsOf(user);
    uint256 lockedDeposits = lockedDepositsOf(user);
    if (amount <= lockedDeposits) {
        return;
    }
    amount = amount.sub(lockedDeposits);
    require(amount <= deposits, "Ankr Deposit#_addNewLockToUser: Insufficient funds");

    require(getConfig(_lockEndsAt_, lockId) == 0, "Ankr Deposit#_addNewLockToUser: Cannot set same lock id");
    // set ends at property for lock
    setConfig(_lockEndsAt_, lockId, endsAt);
    // set amount property for lock
    setConfig(_lockAmount_, lockId, amount);
    setConfig(_lockTotal_, user, getConfig(_lockTotal_, user).add(amount));

    // set lock id
    _userLocks[user].push(lockId);
}
```

Figure 14 The source code of *_addNewLockToUser* function(fixed)

# 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.

| Impact / Likelihood | Severe | High | Medium | Low |
|---|---|---|---|---|
| **Probable** | Critical | High | Medium | Low |
| **Possible** | High | High | Medium | Low |
| **Unlikely** | Medium | Medium | Low | Info |
| **Rare** | Low | Low | Info | Info |

### 3.1.2 Degree of impact

- **Severe**

Severe 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.

### 3.1.4 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.

### 3.1.5 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 the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

**Official Website**

https://www.beosin.com

**Telegram**

https://t.me/+dD8Bnqd133RmNWNl

**Twitter**

https://twitter.com/Beosin_com

**Email**

Contact@beosin.com