BEOSIN
Blockchain Security

ApeironSanctury

Smart Contract Security Audit

No. 202312071600

Dec 7th, 2023

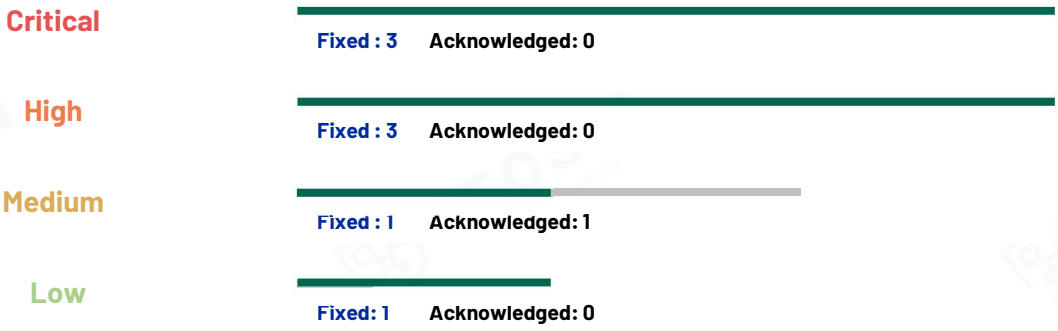
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 3 Critical-risk,3 High-risk, 2 Medium-risk, 1 Low-risk were identified in the ApeironSanctury project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



- **Risk Description:**

1. The project is upgraded by using UUPS agent, please keep the private key in the project side to avoid the leakage of the private key leading to the modification of the realization contract.
2. Since ApeironSanctury-08 is not repaired, hackers can manipulate random numbers and generate NFTs with high-quality properties.

- **Project Description:**

Basic Token Information

Token name	APEA
Token symbol	ApeironApostle
Pre-mint	Origin Token Id from 1001-4600 Zombie Token Id from 4601-4900 Other Token Id from 10001
Total supply	Total volume not constant
Token type	ERC-721

Table 1 APEA token info

Business overview

ApeironApostle is an ERC-721 contract; The main function of ApeironApostleSeasonMinting and ApeironApostleSeasonMintingCaller is to sell NFTs, which can be purchased by users with a signature; The ApeironApostleTicketMinting contract is used to exchange a user's ticket for an NFT, which destroys the user's ticket before the NFT can be minted;The BornPlanet and BreedPlanet contracts implement a planet's born and breed, and the planet relies on VRF's random numbers to compute the planet's attributes when performing the born and breed.

1 Overview

1.1 Project Overview

Project Name	ApeironSanctury
Project language	Solidity
Platform	Ronin chain/Poly chain etc
GitHub	https://github.com/FoonieMagus/ApeironSancturyContract/tree/dev/brian/ronin_migration
Commit Hash	ddade5a9ff80c7cc40c8873a28e3c6f9f8d22cff(Initial) 38ab98ae24bd31fffeb52aaa3bf67e048e6220c1(Final)
Audit scope	ApeironApostle.sol ApeironApostleSeasonMinting.sol ApeironApostleSeasonMintingCaller.sol ApeironApostleTicketMinting.sol ApostleMeta.sol BornPlanet.sol BreedPlanet.sol BreedPlanetBase.sol BreedPlanetData.sol PlanetAttributeManager.sol contracts/utils/ERC721Nonce.sol contracts/utils/IERC721State.sol

1.2 Audit Overview

Audit work duration: Nov 9, 2023 – Dec 7, 2023

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 method 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
ApeironSanctury-01	OpenZeppelin Contracts vulnerable to ECDSA signature malleability	Low	Fixed
ApeironSanctury-02	TicketMinting contracts can be drained of fees	High	Fixed
ApeironSanctury-03	BornPlanet contracts can be drained of fees	High	Fixed
ApeironSanctury-04	Missing mint function for ORIGINAL	High	Fixed
ApeironSanctury-05	The requestBornWithAddress function lacks permission control	Medium	Fixed
ApeironSanctury-06	TicketMinting contract has a random number manipulation vulnerability	Critical	Fixed
ApeironSanctury-07	BornPlanet contract has a random number manipulation vulnerability	Critical	Fixed
ApeironSanctury-08	BreedPlanet contract has a random number manipulation vulnerability	Medium	Acknowledged
ApeironSanctury-09	NFT will be minted for free without limit	Critical	Fixed

Finding Details:

[ApeironSanctury-01] OpenZeppelin Contracts vulnerable to ECDSA signature malleability

Severity Level	Low
Type	General Vulnerability
Lines	ApeironApostleSeasonMintingCaller.sol
Description	Because the project is using OpenZeppelin version 4.5.0, it will be subject to ECDSA malleability attacks.
Recommendation	It is recommended to use the latest version of OpenZeppelin.
Status	**Fixed.** The project has modified the OpenZeppelin version to 4.7.3.

[ApeironSanctury-02] TicketMinting contracts can be drained of fees

Severity Level	High
Type	Business Security
Lines	ApeironApostleTicketMinting.sol#224-310
Description	There is no restriction on user-input data in the `requestTicketMinting` function. This could allow users to make multiple requests with the same data. Since the contract incurs fees when requesting random numbers from the Ronin VRF, attackers could maliciously request to deplete the funds in the contract.

```
    function requestTicketMinting(
        address[] memory _ticketContractAddressArray,
        uint256[] memory _ticketIdArray,
        uint256[] memory _ticketCountArray
    ) external returns (bytes32) {
        // init ticketMintingStructs
        uint256 ticketCount;
        for (uint256 i = 0; i < _ticketCountArray.length; i++) {...}
        // create ticketMintingStruct
        uint256 currentCount = 0;
        for (uint256 i = 0; i < _ticketContractAddressArray.length;
i++) {...}
        // uint256 requestId = requestRandomWords();
        bytes32 requestId = _requestRandomness(
            ronToUseInRandomness,
            address(this)
        );
        // save the request info
        TicketMintingRequestStructMap[requestId].userAddress =
msg.sender;
        // hardhat do not support direct assign struct array
```

```
        for (uint256 i = 0; i < ticketMintingStructs.length; i++) {
            TicketMintingRequestStructMap[requestId].ticketMinting
Structs.push(
                ticketMintingStructs[i]
            );
        }
        TicketMintingRequestStructMap[requestId].isDone = false;
        // emit event
        emit RequestTicketMinting(requestId);
        return requestId;
    }
```

Recommendation It is recommended that users pay a fee for requesting a VRF contract.

Status **Fixed.** The project has changed the code so that the user pays the handling fee.

```
        _requireArgument(
            msg.value >= ronToUseInRandomness,
            "Insufficient ron fee"
        );
```

[ApeironSanctury-03] BornPlanet contracts can be drained of fees

Severity Level	High
Type	Business Security
Lines	BornPlanet.sol#64-98
Description	There is no restriction on user-input data in the `requestMultiBorn` function. This could allow users to make multiple requests with the same data. Since the contract incurs fees when requesting random numbers from the Ronin VRF, attackers could maliciously request to deplete the funds in the contract.

```
    function requestMultiBorn(uint256[] memory planetIdArray)
external {

        uint256[] memory successPlanetArray = new uint256[](

            planetIdArray.length

        );

        uint256[] memory failPlanetArray = new
uint256[](planetIdArray.length);

        string[] memory failReasonArray = new
string[](planetIdArray.length);

        uint256 successCounter = 0;

        uint256 failCounter = 0;

        // loop for each born

        for (uint256 i = 0; i < planetIdArray.length; i++) {

            // require planet is owned by msg.sender

            require(

                planetContract.ownerOf(planetIdArray[i]) ==
msg.sender,

                "Planet is not owned"

            );

            userApprovedBornPlanet[planetIdArray[i]] = msg.sender;

            // try catch can only used in external function

            // using this.requestBornWithAddress msg.sender will
become contract address, so we need to pass userAddress
```

```
            try this.requestBornWithAddress(msg.sender,
planetIdArray[i]) {
                successPlanetArray[successCounter] =
planetIdArray[i];
                successCounter++;
            } catch Error(string memory reason) {
                failPlanetArray[failCounter] = planetIdArray[i];
                failReasonArray[failCounter] = reason;
                failCounter++;
            }
        }
        emit RequestMultiBornSummary(
            msg.sender,
            successPlanetArray,
            failPlanetArray,
            failReasonArray
        );
    }
```

Recommendation It is recommended that users pay a fee for requesting a VRF contract.

Status **Fixed.** The project has changed the code so that the user pays the handling fee.

```
        require(
            msg.value >= ronToUseInRandomness *
planetIdArray.length,
            "Insufficient ron fee"
        );
```

[ApeironSanctury-04] BreedPlanet contracts can be drained of fees

Severity Level	High
Type	Business Security
Lines	BreedPlanet.sol#144-210
Description	There is no restriction on user-input data in the `requestBreedWithAnimus` function. This could allow users to make multiple requests with the same data. Since the contract incurs fees when requesting random numbers from the Ronin VRF, attackers could maliciously request to deplete the funds in the contract.

```
    function requestBreedWithAnimus(
        uint256 planetAId,
        uint256 planetBId,
        uint256 animusUse,
        bool shouldUseMiniBlackhole,
        uint256 time,
        bytes memory signature
    ) external returns (bytes32) {
        bytes32 hash = keccak256(abi.encodePacked(msg.sender,
animusUse, time));
        _requireArgument(
            hash.toEthSignedMessageHash().recover(signature) ==
systemAddress &&
                time + 10 minutes >= block.timestamp, //valid
signature period is 10 minutes
            "Invalid signature"
        );
        return
            _requestBreed(
                planetAId,
                planetBId,
```

```
            animusUse,
            shouldUseMiniBlackhole
        );
    }
    function _requestBreed(
        uint256 planetAId,
        uint256 planetBId,
        uint256 animusUse,
        bool shouldUseMiniBlackhole
    ) internal returns (bytes32) {
        // dry run for check can breed
        _breed(
            msg.sender,
            planetAId,
            planetBId,
            animusUse,
            shouldUseMiniBlackhole,
            true
        );
        // request rng for get random number
        bytes32 requestHash = _requestRandomness(
            ronToUseInRandomness,
            address(this)
        );
        BreedStruct memory breedStruct = BreedStruct(
            msg.sender,
            planetAId,
            planetBId,
```

```
            shouldUseMiniBlackhole,
            false,
            0
        );
        BreedStructMap[requestHash] = breedStruct;
        animusUseMap[requestHash] = animusUse;
        emit RequestBreed(requestHash);
        return requestHash;
    }
```

Recommendation It is recommended that users pay a fee for requesting a VRF contract.

Status **Fixed.** The project has changed the code so that the user pays the handling fee.

```
        _requireArgument(
            msg.value >= ronToUseInRandomness,
            "Insufficient ron fee"
        );
```

[ApeironSanctury-05] The requestBornWithAddress function lacks permission control

Severity Level	Medium
Type	Business Security
Lines	BornPlanet.sol#110-118
Description	Due to the lack of permission restrictions in the `requestBornWithAddress` function, anyone can potentially perform a "born" operation on a user's planet.

```
function requestBornWithAddress(
    address userAddress,
    uint256 planetId
) external returns (bytes32) {
    require(
        userApprovedBornPlanet[planetId] == userAddress,
        "Planet is not approved for born"
    );
    return _requestBorn(userAddress, planetId);
}
```

Recommendation	It is advisable to enhance the permissions for the `requestBornWithAddress` function.
Status	**Fixed.** The project added permissions to the `requestBornWithAddress` function.

```
function requestBornWithAddress(
    address userAddress,
    uint256 planetId
) external returns (bytes32) {
    // check planet is approved for born
    require(
        userApprovedBornPlanet[planetId] == userAddress,
        "Planet is not approved for born"
```

```
        );

        // check caller is this contract

        require(msg.sender == address(this), "Caller is not this
contract");

        return _requestBorn(userAddress, planetId);

    }
```

[ApeironSanctury-06] TicketMinting contract has a random number manipulation vulnerability

Severity Level	**Critical**
Type	Business Security
Lines	ApeironApostleTicketMinting.sol
Description	when users use a ticket to mint an NFT, the user's ticket is not deducted in the `requestTicketMinting` function. Instead, the deduction of the user's ticket occurs after obtaining the random number. Therefore, a hacker could observe the random numbers in the memory pool of the VRF contract during the transaction initiation. If the NFT properties generated by the random number are unfavorable, the hacker could prematurely transfer the ticket, causing the callback failure of the Ronin VRF contract, until generated satisfactory NFT.
Recommendation	It is recommended that when a user requests a random number, the contract deducts the user's ticket.
Status	**Fixed.** The project side destroys the user's ticket when it makes the request.

```
            ERC1155Burnable(_ticketContractAddressArray[i]).burn(
                msg.sender, // burt from
                _ticketIdArray[i], // burt ticket id
                _ticketCountArray[i] // burn ticket count
            );
```

[ApeironSanctury-07] BornPlanet contract has a random number manipulation vulnerability

Severity Level	**Critical**
Type	Business Security
Lines	BornPlanet.sol#160-188
Description	The `_born` function checks the ownership of the planetId when the user is doing a BORN on the NFT. If a hacker observes an unsatisfactory random number generated by a VRF contract in the memory pool, the hacker can move the NFT corresponding to planetId away, causing the Ronin VRF contract callback to fail until a satisfactory NFT is born.

```
    function _born(
        address userAddress,
        uint256 planetId,
        bool isDryRun // if isDryRun, not updatePlanetData
    ) internal {
        IApeironPlanet.PlanetData memory planetData =
_getPlanetData(planetId);
        // check can born
        require(
            planetContract.ownerOf(planetId) == userAddress,
            "Planet is not owned"
        );
        require(planetData.bornTime == 0, "Planet already born");
        require(_hasParent(planetId), "Planet has no parent");
        require(
            breedPlanetDataContract.getPlanetNextBornTime(planetId
) <
                block.timestamp,
            "Born time is pass for planetNextBornMap time"
        );
```

```
        if (!isDryRun) {
            // update planet.gene
            uint256 geneId = _convertToGeneId(
                _updateAttributesOnBorn(planetId)
            );
            // update planet as borned
            planetContract.updatePlanetData(planetId, geneId, 0, 0,
3, true);
            emit BornSuccess(planetId);
        }
    }
```

Recommendation When a VRF contract callback is recommended, users do not have the right to refuse.

Status **Fixed.** The project does not check the ownership of the planet during the callback, and will perform born operations on the NFT even if the NFT is transferred.

```
        if (isDryRun) {
            // only check when _requestBorn, not check when
_fulfillRandomSeed
            require(
                planetContract.ownerOf(planetId) == userAddress,
                "Planet is not owned"
            );
        }
```

[ApeironSanctury-08] BreedPlanet contract has a random number manipulation vulnerability

Severity Level	**Medium**
Type	Business Security
Lines	BreedPlanet.sol
Description	When a user breeds on an NFT, the `breed` function checks whether the user's fee is sufficient. If a hacker observes in the memory pool that the random number generated by the VRF contract is unsatisfactory, the hacker can divert the handling fee required for breed, causing the Ronin VRF contract callback to fail until a satisfactory NFT is generated.
Recommendation	When a VRF contract callback is recommended, users do not have the right to refuse.
Status	**Acknowledged.** Description of the project side: Because NFT in breed, mainly rely on the attributes of the parents, the influence of random numbers is not very big, so the code is not modified.

[ApeironSanctury-09] NFT will be minted for free without limit

Severity Level	**Critical**
Type	Business Security
Lines	ApeironApostleSeasonMinting.sol#356-404
Description	In the `_purchase` function, since the NFT minting is ahead of the book update and the starting entry function has no re-reentry check, it will result in unlimited free minting of the NFT.

```
    function _purchase(
        address _user,
        SEASON_MINT_TYPE _mintType,
        ApostleMeta.ApostleClass _apostleClass,
        uint256 _dungeonApostleId,
        uint256 _gene,
        uint256 _iv,
        uint256 _price
    ) internal {
        bool isFreeMint = (_price == 0);
        // transfer token if this is not free mint
        if (!isFreeMint) {...}
        // mint NFT
        uint256 tokenId = apostleContract.safeMint(
            _gene,
            _iv,
            ApeironApostle.MINT_TYPE.TYPE_OTHER, // mint type is always TYPE_OTHER
            _user
        );
        // update free mint count
        if (isFreeMint) {
```

```
            addressFreeMintedMapping[_mintType][_user] += 1;
        }
        // update address minted class count
        addressMintedClassMapping[_mintType][_user][
            uint256(_apostleClass)
        ] += 1;
        // update minted dungeon apostle id
        mintedDungeonApostleIdMapping[_dungeonApostleId] =
tokenId;
```

Recommendation

1. Add reentry check.
2. The books are updated first, and then transfers are made.

Status

Fixed. The project added reentrant check and transfer sequence.

```
        if (isFreeMint) {
            addressFreeMintedMapping[_mintType][_user] += 1;
        }
        // update address minted class count
        addressMintedClassMapping[_mintType][_user][
            uint256(_apostleClass)
        ] += 1;
        // mint NFT
        uint256 tokenId = apostleContract.safeMint(
            _gene,
            _iv,
            ApeironApostle.MINT_TYPE.TYPE_OTHER, // mint type is
always TYPE_OTHER
            _user
        );
        mintedDungeonApostleIdMapping[_dungeonApostleId] =
tokenId;
```

2 Appendix

2.1 Vulnerability Assessment Metrics and Status in Smart Contracts

2.1.1 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

2.1.2 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.

2.1.4 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.

2.1.5 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.

2.2 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.

2.3 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.

2.4 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/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com