BEOSIN
Blockchain Security

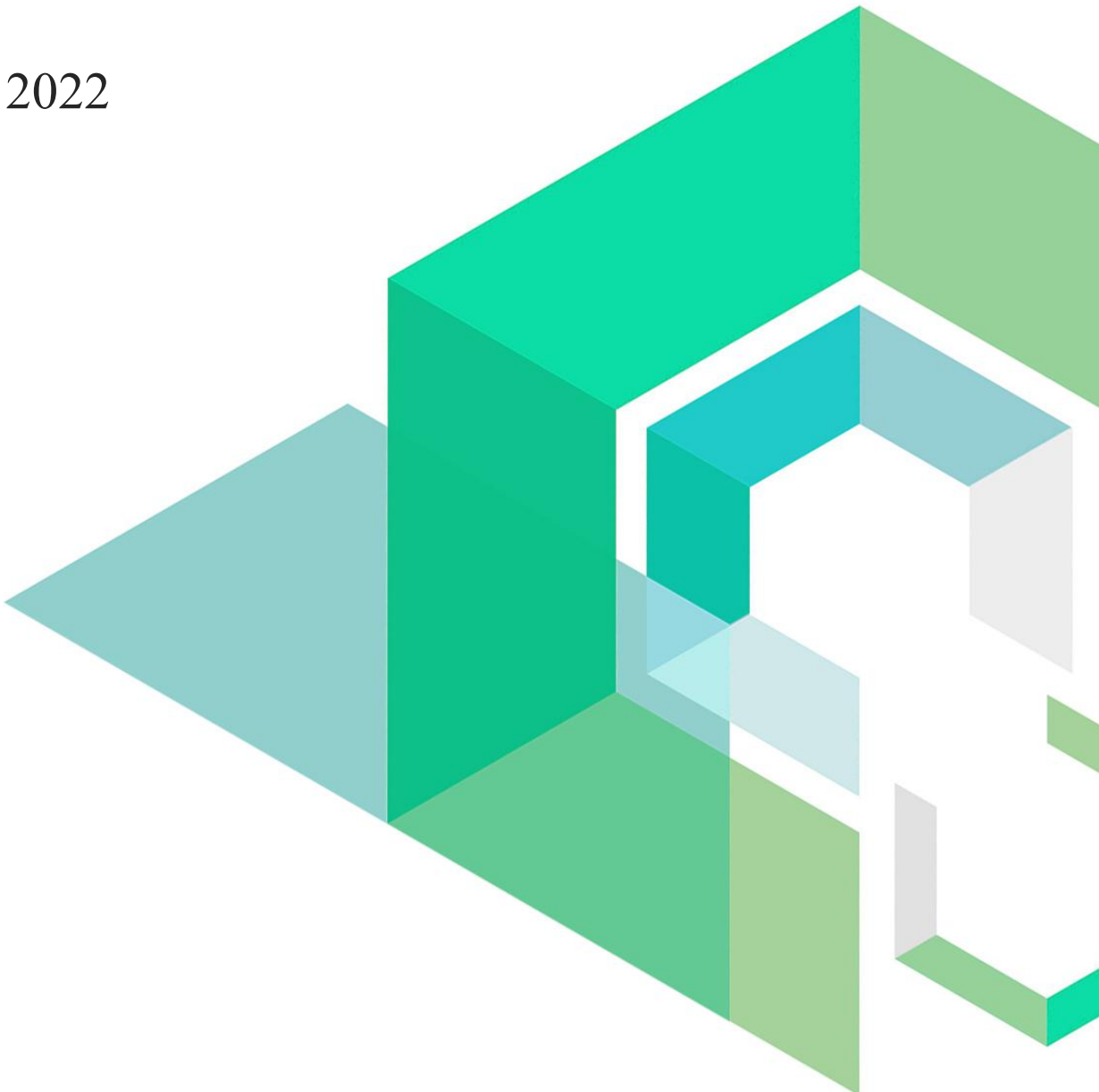
Big Cake

Smart Contract Security Audit

V1.0

No. 202209291804

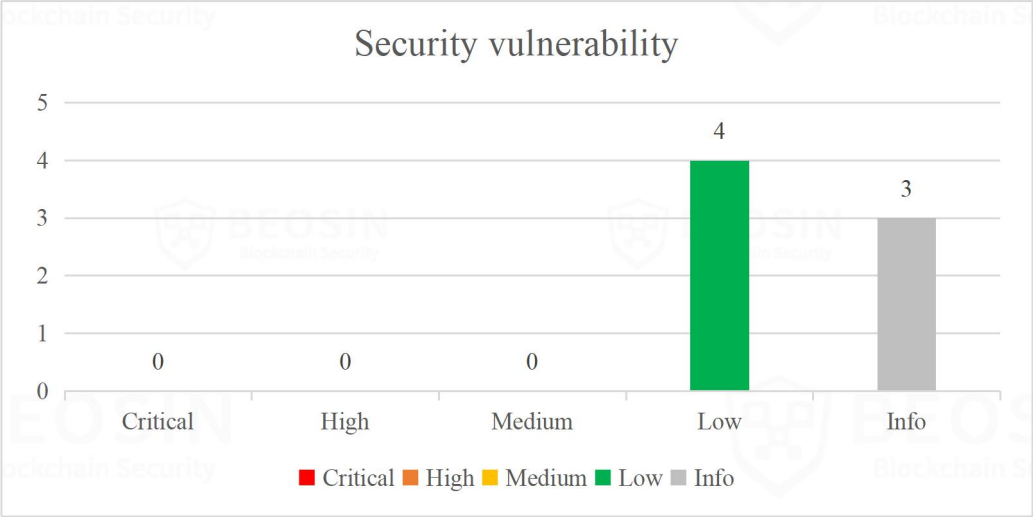
Sep 29th, 2022



Contents

Summary of Audit Results

After auditing, 4 Low-risk items and 3 Info items was identified in the Big Cake 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 audit, in this project, if the address settings of the staked tokens and reward tokens are confused, it may lead to the distribution of some users' staked assets as rewards. Under special circumstances, users may not be able to withdraw the staked assets normally due to an error in the calculation of dividends (but they can call *emergencyWithdraw* function to withdraw assets). It is recommended that users pay attention to the relevant settings of the stake pool when participating in the project.

- **Project Description:**

1. Business overview

Big Cake is a staking reward project on BNB Chain. In the contract, the deployer can add different types of staking mining pools. The pool with Pid=0 can receive additional dividends. As of the completion of the audit, there are only staking pools with Pid=0. Using Pancake LPs(BSC-USD-B-CAKE 5) tokens as staking tokens and BIG CAKE as reward tokens. The deployer can set the time of the staking pool, the reward amount of each block, the total amount of rewards, the address of reward token, and the reward for inviters, among which the address of the staked token can be changed but requires the staking pool fund to be 0. New users can bind an inviter, and after having a stake record, they can distribute a part of the reward to the inviter.

1 Overview

1.1 Project Overview

Project Name	Big Cake
Platform	BNB Chain
Contract Address	0x311bA0AB0F8be16695250B375824f53e4186B295

1.2 Audit Overview

Audit work duration: Sep 26, 2022 – Sep 29, 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
Big Cake-1	*AddPool* function can add the same address	Low	Acknowledged
Big Cake-2	The reward token address can be set as the staking token address	Low	Acknowledged
Big Cake-3	Reward-related function call failure risk	Low	Acknowledged
Big Cake-4	PendingReward calculation error	Low	Acknowledged
Big Cake-5	Query interface risk	Info	Acknowledged
Big Cake-6	Unlimited cost settings	Info	Acknowledged
Big Cake-7	Missing event trigger	Info	Acknowledged

Status Notes:

- Big Cake-1 is unfixed, when the staking token address of the staking pool and the reward token address are set to the same address, it will cause the user to withdraw the principal when withdrawing the reward.
- Big Cake-2 is unfixed, if the current staked tokens are replaced with reward tokens in other staked pools, it will cause staked tokens to be issued to other pools as rewards.
- Big Cake-3 is unfixed, when the reward in the staking pool is insufficient, the failure to issue the reward will cause the basic function call to fail.
- Big Cake-4 is unfixed, in the staking pool with pid=0, because the pendingReward variable is only updated in the deposit function, if withdraw is called multiple times, the pendingReward variable may be incorrectly calculated.
- Big Cake-5 is unfixed does not cause any risk.
- Big Cake-6 is unfixed, if _inviterFee is set too high, the inviter will get a high reward.
- Big Cake-7 is unfixed does not cause any risk.

Finding Details:

[Big Cake-1] *AddPool* function can add the same address

Severity Level	Low
Type	Business Security
Lines	BCakeMintPool.sol #L191-220
Description	When adding a pool, if the reward token and the Staking token are set to the same address, the user may receive the principal when claiming the reward.

```
191     function addPool(
192         address lpToken,
193         address rewardToken,
194         uint256 rewardPerBlock,
195         uint256 startTime,
196         uint256 endTime,
197         uint256 timePerBlock,
198         uint256 totalReward
199     ) external onlyOwner {
200         uint256 blockTimestamp = block.timestamp;
201         uint256 blockNum = block.number;
202         uint256 startBlock;
203 >       if (startTime > blockTimestamp) {
205 >       } else {···
207         }
208         _poolInfos.push(PoolInfo({
```

Figure 1 Source code of *addPool* function

Recommendations	It is recommended to add the judgment that the staked token cannot be the same as the reward token.
Status	Acknowledged.

[Big Cake-2] The reward token address can be set as the staking token address

Severity Level	**Low**
Type	Business Security
Lines	BCakeMintPool.sol #L238-241
Description	When adding a staking pool or changing the token address of the staking pool, the address of the reward token can be set as the address of the staking token, and there is no state restriction. If the current staking token is set as the reward token in other staking pools, the staking tokens in the current pool will be issued as rewards in other pools.

```
238     function setPoolRewardToken(uint256 pid, address token) external onlyOwner {
239         PoolInfo storage pool = _poolInfos[pid];
240         pool.rewardToken = token;
241     }
```

Figure 2 Source code of *setPoolRewardToken* function

Recommendations	It is recommended to add the judgment that the reward token address cannot be set as the Staking token address.
Status	Acknowledged.

[Big Cake-3] Reward-related function call failure risk

Severity Level	**Low**
Type	Business Security
Lines	BCakeMintPool.sol #L300-322, L467-473, L490-500
Description	If the reward token balance in the contract is insufficient, the user reward will not be issued, and it will cause the function call to fail. In the *claimToken* function, the owner can arbitrarily transfer the excess rewards. In the *_claimDividend* function, the function call of normal function may also fail due to insufficient reward.

```
300  function _claim(uint256 pid, UserInfo storage user, address account) private {
301      if (0 == pid) {
302          _claimDividend(account);
303      }
304      PoolInfo storage pool = _poolInfos[pid];
305      uint256 userAmount = user.amount;
306      if (userAmount > 0) {
307          uint256 pendingAmount = userAmount * pool.accPerShare / _rewardFactor - user.rewardDebt;
308          if (pendingAmount > 0) {
309              user.rewardDebt = userAmount * pool.accPerShare / _rewardFactor;
310              IERC20 rewardToken = IERC20(pool.rewardToken);
311              require(rewardToken.balanceOf(address(this)) >= pendingAmount, "rewardToken not enough");
312              address invitor = _invitor[account];
313              if (address(0) != invitor) {
314                  uint256 inviteAmount = pendingAmount * _inviteFee / _feeDivFactor;
315                  if (inviteAmount > 0) {
316                      pendingAmount -= inviteAmount;
317                      rewardToken.transfer(invitor, inviteAmount);
318                  }
319              }
320              rewardToken.transfer(account, pendingAmount);
321          }
322      }
```

Figure 3 Source code of *_claim* function

```
467  function claimToken(address token, address to, uint256 amount) external onlyOwner {
468      uint256 maxClaim = IERC20(token).balanceOf(address(this)) - _poolLpBalances[token];
469      if (amount > maxClaim) {
470          amount = maxClaim;
471      }
472      IERC20(token).transfer(to, amount);
473  }
```

Figure 4 Source code of *claimToken* function

```
490  function _claimDividend(address account) private {
491      UserInfo storage userInfo = _userInfos[0][account];
492      if (userInfo.amount > 0) {
493          uint256 accReward = userInfo.amount * _accDividendRewardPerShare / _rewardFactor;
494          uint256 pendingReward = accReward - _dividendRewardDebt[account];
495          if (pendingReward > 0) {
496              _dividendRewardDebt[account] = accReward;
497              IERC20(_dividendRewardToken).transfer(account, pendingReward);
498          }
499      }
500  }
```

Figure 5 Source code of *_claimDividend* function

Recommendations	It is recommended to judge whether the number of reward tokens is sufficient before issuing rewards.
Status	Acknowledged.

[Big Cake-4] PendingReward calculation error

Severity Level	**Low**
Type	Business Security
Lines	BCakeMintPool.sol #L490-500
Description	The value of the _dividendRewardDebt variable is updated only when the *deposit* function is called, and the _dividendRewardDebt variable in the *_claimDividend* function is not updated in the *withdraw* function. When the user calls the *withdraw* function, if pendingReward is 0, then the user's _dividendRewardDebt will not be updated, but the corresponding staking amount will be reduced, which will cause the function call to fail due to an abnormal pendingReward calculation when the user withdraws next time.

```
162         if (0 == pid) {
163             uint256 accReward = user.amount * _accDividendRewardPerShare / _rewardFactor;
164             _dividendRewardDebt[account] = accReward;
165         }
166     }
```

Figure 6 Source code of dividendRewardDebt

```
490     function _claimDividend(address account) private {
491         UserInfo storage userInfo = _userInfos[0][account];
492         if (userInfo.amount > 0) {
493             uint256 accReward = userInfo.amount * _accDividendRewardPerShare / _rewardFactor;
494             uint256 pendingReward = accReward - _dividendRewardDebt[account];
495             if (pendingReward > 0) {
496                 _dividendRewardDebt[account] = accReward;
497                 IERC20(_dividendRewardToken).transfer(account, pendingReward);
498             }
499         }
500     }
```

Figure 7 Source code of *_claimDividend* function

Recommendations	It is recommended to calculate pendingReward with latest _dividendRewardDebt after calling *withdraw* function.
Status	Acknowledged.

[Big Cake-5] Query interface risk

Severity Level	Info
Type	Coding Conventions
Lines	BCakeMintPool.sol #L391-398
Description	In the *getPoolInfo* function, the *getTokenPrice* function is used to calculate the price. The price depends on the quantity ratio of USDT tokens to staking tokens in the pair contract. If the price of this interface is used for calculation, it may be subject to flash loan price manipulation attacks.

```
391    function getTokenPrice(address token) public view returns (uint256 price){
392        address usdtPair = _factory.getPair(token, _usdtAddress);
393        uint256 usdtAmount = IERC20(_usdtAddress).balanceOf(usdtPair);
394        uint256 tokenAmount = IERC20(token).balanceOf(usdtPair);
395        if (tokenAmount > 0) {
396            price = 10 ** IERC20(token).decimals() * usdtAmount / tokenAmount;
397        }
398    }
```

Figure 8 Source code of *getTokenPrice* function

Recommendations	It is recommended not to use this interface outside of queries.
Status	Acknowledged.

[Big Cake-6] Unlimited cost settings

Severity Level	Info
Type	Business Security
Lines	BCakeMintPool.sol #L475-477
Description	There is no upper limit on the fee setting, and the bound inviter may receive excessive rewards.

```
475     function setInviteFee(uint256 fee) external onlyOwner
476         _inviteFee = fee;
477     }
```

Figure 9 Source code of *setInviteFee* function

Recommendations	It is recommended to add scope restrictions to the _inviteFee variable.
Status	Acknowledged.

[Big Cake-7] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	BCakeMintPool.sol #L525-526, L222-225, L227-230, L232-236, L238-241, L479-487, L475-477
Description	Important variable changes. The function is missing an event trigger.

```
525     function setPoolAdmin(address adr, bool enable) external onlyOwner {
526         _poolAdmin[adr] = enable;
527     }
```

```
222     function setPoolRewardPerBlock(uint256 pid, uint256 rewardPerBlock) external onlyOwner {
223         _updatePool(pid);
224         _poolInfos[pid].rewardPerBlock = rewardPerBlock;
225     }
```

```
227     function setPoolTotalReward(uint256 pid, uint256 totalReward) external onlyOwner {
228         _updatePool(pid);
229         _poolInfos[pid].totalReward = totalReward;
230     }
```

```
232     function setPoolLP(uint256 pid, address lp) external onlyOwner {
233         PoolInfo storage pool = _poolInfos[pid];
234         require(pool.totalAmount == 0, "started");
235         pool.lpToken = lp;
236     }
```

```
238     function setPoolRewardToken(uint256 pid, address token) external onlyOwner {
239         PoolInfo storage pool = _poolInfos[pid];
240         pool.rewardToken = token;
241     }
```

```
479     function setUsdtAddress(address usdt) external onlyOwner {
480         _usdtAddress = usdt;
481     }
482
483     function setSwapRouter(address swapRouterAddress) external onlyOwner {
484         ISwapRouter swapRouter = ISwapRouter(swapRouterAddress);
485         _factory = ISwapFactory(swapRouter.factory());
486         _swapRouter = swapRouter;
487     }
```

```
475     function setInviteFee(uint256 fee) external onlyOwner {
476         _inviteFee = fee;
477     }
```

Figure 10 Source code of *setPoolAdmin, setPoolRewardPerBlock, setPoolTotalReward, setPoolLP, setPoolRewardToken, setUSDTAddress, setSwapRouter, setInviteFee* functions

Recommendations	It is recommended to add event triggers.
Status	Acknowledged.

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