BEOSIN
Blockchain Security

Kinghash

Smart Contract Security Audit

V1.0

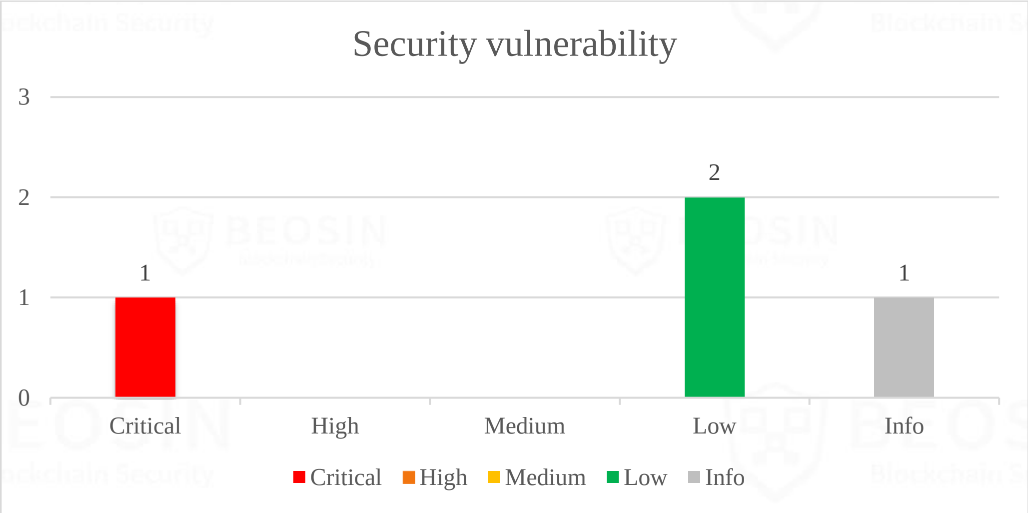
No. 202210251808

Oct 25th, 2022

Contents

Summary of audit results

After auditing, 1 Critical-risk, 2 Low-risk and 1 Info-risk items were identified in the Kinghash 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. For the project party, it is necessary to use multi-signature wallet, TimeLock contract, or DAO, etc. as each contracts' owner, and carefully manage aggregator permissions; For the user, it is necessary to pay attention to the fund security risks due to centralization risks.

2. This audit report is only for the current code, but the business logic contract of the current project is upgradable, and the code after the upgrade cannot be determined. After the upgrade, the risk of capital and data loss may be introduced. Users should pay attention to the related risks when interacting with the upgraded contract.

- **Project Description:**

1. Business overview

The Kinghash project includes an Aggregator contract, a NodeRewardVault contract, a NodeCapitalVault contract and a ValidatorNft contract. After each contract is deployed, the deployer is granted Owner authority. The owner can transfer authority to other address such as TimeLockController for permission control and renounce the owner authority of the contract when transferred to address zero.

In the Aggregator contract, users can use the *stake* function to make eth staking. After the staking, users can get the corresponding token as the certificate of the verifier and pay 32 ETH during the staking. Each public key corresponds to a token certificate, but each address can hold multiple tokens. Users can also choose the strategy of staking to Lido or Rocketpool platform through the *stake* function. Users can use the *stake* function to trade tokens, among this, the “Authority” acts as an oracle, providing off-chain data and signature functions to smart contracts. This contract has a pause function, users cannot call the *stake* function for various routing transactions when a contract is suspended. At present, the contract does not support users to withdraw their staking assets. The *unstake* function will be implemented after the Ethereum Shanghai upgrade according to the project party.

In the NodeRewardVault contract, the main function is to store the staking reward fund and calculate the staking reward. The owner can set the Dao address, authority address and Aggregator address, as well as the “comission” rate and tax rate during the transaction, and the Aggregator contract can call related functions of this contract to claim rewards. At present, there is no reward source for this project. The project party said that after the Shanghai upgrade, the ETH staking reward will be claimed to this contract for reward distribution.

In the ValidatorNft contract, the initial supply of tokens is zero and the maximum supply of tokens is “6,942,069,420”, which can be minted and burned only by the specified Aggregator contract, currently the burn interface is not implemented in Aggregator and will be updated after Ethereum Shanghai upgrade according to the project party's instructions. The contract overwrites the *isApprovedForAll* function. The Aggregator address can directly transfer any token without authorization, and the OpenSeaProxy address can do the same when the OpenSeaProxy is active. The owner can switch the activation status of OpenSeaProxy. Users can claim the rewards of staking manually with their tokens, or automatically claim when they transfer tokens.

The NodeCapitalVault contract is used to receive the principal for staking, and will be upgraded after the Shanghai upgrade according to the project party.

1 Overview

1.1 Project Overview

Project Name	Kinghash
Platform	Ethereum
Audit scope	https://github.com/King-Hash-Org/KingHashAggregator
Commit Hash	80818720b4980a27b7ee4ce6e79f86224e062167 (Unfixed) ad4530fe7d756be7f41460c1b00f5676a0ad4462 (Fixed)

1.2 Audit Overview

Audit work duration: September 19, 2022 – October 25, 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
Kinghash-1	User’s funds may be locked up in contract	**Critical**	Fixed
Kinghash-2	Records are not updated after burning tokens	**Low**	Fixed
Kinghash-3	Risk of centralization	**Low**	Partially Fixed
Kinghash-4	Missing event trigger	**Info**	Fixed

Status Notes:

- Kinghash-3 is partially fixed and the owner of the contract in this project has high control authority and can change key parameters in the project, which has a certain centralization risk. Misoperation or loss of the owner's private key may result in loss of user assets.

[Kinghash-1] User's funds may be locked up in contract

Severity Level Critical

Type Business Security

Lines ValidatorNftRouter.sol #L73-98

Description In the *_tradeRoute* function of the contract, the seller's token is transferred to the buyer, but the amount paid by the buyer is not transferred to the seller. Moreover, there is no relevant withdrawal function in the Aggregator contract, which may lead to a large amount of funds locked in the contract and unable to withdraw.

```
73    function _tradeRoute(Trade memory trade) private returns (uint256) {
74        require(trade.expiredHeight > block.number, "Trade has expired");
75
76        // change this in the future
77        uint256 sum = 0;
78        uint256 i = 0;
79        for (i = 0; i < trade.prices.length; i++) {
80            sum += trade.prices[i];
81        }
82
83        // change this in the future
84        bytes32 masterHash;
85        for (i = 0; i < trade.userListings.length; i++) {
86            UserListing memory userListing = trade.userListings[i];
87            bytes32 hash = keccak256(abi.encodePacked(userListing.tokenId, userListing.rebate, userListing.expiredHeight));
88            signercheck(userListing.signature.s, userListing.signature.r, userListing.signature.v, hash, userListing.signature.signer);
89
90            masterHash = keccak256(abi.encodePacked(hash, masterHash));
91            nftContract.safeTransferFrom(nftContract.ownerOf(userListing.tokenId), trade.receiver, userListing.tokenId);
92        }
93
94        masterHash = keccak256(abi.encodePacked(trade.prices, trade.expiredHeight, trade.receiver, masterHash));
95        signercheck(trade.signature.s, trade.signature.r, trade.signature.v, masterHash, vault.authority());
96
97        return sum;
98    }
```

Figure 1 Source code of *_tradeRoute* function (Unfixed)

Recommendations It is recommended to transfer the ETH paid by buyer to the seller.

Status Fixed. The project party has added related processing logic.

```
function _tradeRoute(bytes calldata data) private returns (uint256) {
    require(address(bytes20(data[12:32])) == msg.sender, "Not allowed to make this trade");
    require(uint256(bytes32(data[96:128])) > block.number, "Trade has expired");

    uint256 sum = 0;
    uint256 i = 0;

    for (i = 0; i < uint256(bytes32(data[128:160])); i++) {
        uint256 price = uint256(bytes32(data[160 + i * 224:192 + i * 224]));
        uint256 tokenId = uint256(bytes32(data[192 + i * 224:224 + i * 224]));
        uint256 rebate = uint256(bytes32(data[224 + i * 224:256 + i * 224]));
        uint256 expiredHeight = uint256(bytes32(data[256 + i * 224:288 + i * 224]));
        address signer = address(bytes20(data[352 + i * 224:372 + i * 224]));
        uint64 nonce = uint64(bytes8(data[376 + i * 224:384 + i * 224]));

        require(expiredHeight > block.number, "Listing has expired");
        require(nftContract.ownerOf(tokenId) == signer, "Not owner");
        require(nonce == nonces[tokenId], "Incorrect nonce");

        nonces[tokenId]++;
        sum += price;

        bytes32 hash = keccak256(abi.encodePacked(tokenId, rebate, expiredHeight, nonce));
        signercheck(bytes32(data[320 + i * 224:352 + i * 224]), bytes32(data[288 + i * 224:320 + i * 224]), uint8(bytes1(data[372 + i * 224])), hash, signer);

        uint256 nodeCapital = nftContract.nodeCapitalOf(tokenId);
        uint256 userPrice = price;
        if (price > nodeCapital) {
            userPrice = price - (price - nodeCapital) * vault.comission() / 10000;
            payable(vault.dao()).transfer(price - userPrice);
        }
        require(userPrice > 30 ether, "Node too cheap");

        payable(signer).transfer(userPrice);
        nftContract.safeTransferFrom(signer, msg.sender, tokenId);
        nftContract.updateNodeCapital(tokenId, price);

        emit NodeTrade(tokenId, signer, msg.sender, price);
    }

    bytes32 authHash = keccak256(abi.encodePacked(data[160:], uint256(bytes32(data[96:128])), msg.sender));
    signercheck(bytes32(data[64:96]), bytes32(data[32:64]), uint8(bytes1(data[1])), authHash, vault.authority());

    return sum;
}
```

Figure 2 Source code of *_tradeRoute* function (Fixed)

[Kinghash-2] Records are not updated after burning tokens

Severity Level	Low
Type	Business Security
Lines	ValidatorNft.sol #L103-105, L145-L160

Description

The value of *_totalHeight* and *_gasHeight* will not be updated after the *whiteListBurn* function burns the token, which may affect the reward calculation and result in a lower reward for the user.

```
103 function whiteListBurn(uint256 tokenId) external onlyAggregator {
104     _burn(tokenId);
105 }
```

Figure 3 Source code of *whiteListBurn* function

```
function _burn(uint256 tokenId, bool approvalCheck) internal virtual {
    uint256 prevOwnershipPacked = _packedOwnershipOf(tokenId);

    address from = address(uint160(prevOwnershipPacked));

    (uint256 approvedAddressSlot, address approvedAddress) = _getApprovedSlotAndAddress(tokenId);

    if (approvalCheck) {
        // The nested ifs save around 20+ gas over a compound boolean condition.
        if (!_isSenderApprovedOrOwner(approvedAddress, from, _msgSenderERC721A()))
            if (!isApprovedForAll(from, _msgSenderERC721A())) revert TransferCallerNotOwnerNorApproved();
    }

    _beforeTokenTransfers(from, address(0), tokenId, 1);
```

Figure 4 Part of source code of *_burn* function (Unfixed)

```
145 function _beforeTokenTransfers(
146         address from,
147         address to,
148         uint256 startTokenId,
149         uint256 quantity
150     ) internal virtual override
151 {
152     // no need to claim reward if user is burning or minting nft
153     if (from == address(0) || to == address(0)) {
154         return;
155     }
156 
157     for (uint256 i = 0; i < quantity; i++) {
158         _claimRewards(startTokenId + i);
159     }
160 }
```

Figure 5 Source code of *_beforeTokenTransfers* function (Unfixed)

Recommendations

It is recommended to update the corresponding *_totalHeight* and *_gasHeight* value after the tokens are burned.

Status	Fixed. The reward is claimed before burning, and the reward calculation method has been modified, since the length increases with each transfer, a list that is too long over time may cause a gas shortage problem.

```
function _settle() private {
    uint256 outstandingRewards = address(this).balance - unclaimedRewards - daoRewards;
    if (outstandingRewards == 0 || cumArr[cumArr.length - 1].height == block.number) {
        return;
    }

    uint256 daoReward = (outstandingRewards * _comission) / 10000;
    daoRewards += daoReward;
    outstandingRewards -= daoReward;
    unclaimedRewards += outstandingRewards;

    uint256 averageRewards = outstandingRewards / _nftContract.totalSupply();
    uint256 currentValue = cumArr[cumArr.length - 1].value + averageRewards;
    RewardMetadata memory r = RewardMetadata({
        value: currentValue,
        height: block.number
    });
    cumArr.push(r);

    emit Settle(block.number, averageRewards);
}
```

Figure 6 Source code of *_settle* function

[Kinghash-3] Risk of centralization

Severity Level	**Low**
Type	Business Security
Lines	NodeRewardVault #L83-86, ValidatorNft #L88-105, L163-183
Description	The Aggregator can call the *transferFrom* function to transfer any toke and mint any number of tokens for free as long as the total amount does not exceed the upper limit, and the Aggregator also can call the *transfer* function to extract the reward funds in the vault contract, which may involve centralization risks, for example, it may lead to user's tokens being transferred and burned without authorization, less staking rewards or no rewards to be received, etc.

```
83      function transfer(uint256 amount, address to) external override nonReentrant onlyAggregator {
84          require(to != address(0), "Recipient address provided invalid");
85          payable(to).transfer(amount);
86      }
```

Figure 7 Source code of *transfer* functions

```
88    function whiteListMint(bytes calldata pubkey, address _to) external onlyAggregator {
89      require(
90        totalSupply() + 1 <= MAX_SUPPLY,
91        "not enough remaining reserved for auction to support desired mint amount"
92      );
93      require(!validatorRecords[pubkey], "Pub key already in used");
94
95      validatorRecords[pubkey] = true;
96      _validators.push(pubkey);
97      _gasHeights.push(block.number);
98      _totalHeight += block.number;
99
100     _safeMint(_to, 1);
101   }
102
103   function whiteListBurn(uint256 tokenId) external onlyAggregator {
104     _burn(tokenId);
105   }
```

Figure 8 Source code of ValidatorNft contract

```
function isApprovedForAll(address owner, address operator)
    public
    view
    override
    returns (bool)
{
    // Get a reference to OpenSea's proxy registry contract by instantiating
    // the contract using the already existing address.

    if (
        _isOpenSeaProxyActive &&
        OPENSEA_PROXY_ADDRESS == operator
    ) {
        return true;
    }
    if (operator == _aggregatorProxyAddress) {
        return true;
    }

    return super.isApprovedForAll(owner, operator);
}
```

Figure 9 Source code of *isApprovedForAll* function

```
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public payable virtual override {
    uint256 prevOwnershipPacked = _packedOwnershipOf(tokenId);

    if (address(uint160(prevOwnershipPacked)) != from) revert TransferFromIncorrectOwner();

    (uint256 approvedAddressSlot, address approvedAddress) = _getApprovedSlotAndAddress(tokenId);

    // The nested ifs save around 20+ gas over a compound boolean condition.
    if (!_isSenderApprovedOrOwner(approvedAddress, from, _msgSenderERC721A()))
        if (!isApprovedForAll(from, _msgSenderERC721A())) revert TransferCallerNotOwnerNorApproved();

    if (to == address(0)) revert TransferToZeroAddress();

    _beforeTokenTransfers(from, to, tokenId, 1);
```

Figure 10 Part of source code of *transferFrom* function

Recommendations	It is recommended to transfer and burn the token after the user's authorization, and use the TimeLock or DAO mechanism to manage the permissions of the owner and Aggregator address.
Status	Partially fixed. According to the explanation of the project party, the burning token will be updated to require user signature verification in Aggregator contract in the future, and multi-signature and time lock will be used to manage the owner's permission.

```
contract TimelockController is AccessControl, IERC721Receiver, IERC1155Receiver {
    bytes32 public constant TIMELOCK_ADMIN_ROLE = keccak256("TIMELOCK_ADMIN_ROLE");
    bytes32 public constant PROPOSER_ROLE = keccak256("PROPOSER_ROLE");
    bytes32 public constant EXECUTOR_ROLE = keccak256("EXECUTOR_ROLE");
    bytes32 public constant CANCELLER_ROLE = keccak256("CANCELLER_ROLE");
    uint256 internal constant _DONE_TIMESTAMP = uint256(1);
```

Figure 11 Source code of TimeLock contract

[Kinghash-4] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	NodeRewardVault #L83-106, ValidatorNft #L114-125, L187-192
Description	In the NodeRewardVault and ValidatorNft contract, there is no event trigger for following functions.

```
83   function transfer(uint256 amount, address to) external override nonReentrant onlyAggregator {
84       require(to != address(0), "Recipient address provided invalid");
85       payable(to).transfer(amount);
86   }
87
88   function setComission(uint256 comission_) external onlyOwner {
89       require(comission_ < 10000, "Comission cannot be 100%");
90       _comission = comission_;
91   }
92
93   function setDao(address dao_) external onlyOwner {
94       require(dao_ != address(0), "DAO address provided invalid");
95       _dao = dao_;
96   }
97
98   function setAuthority(address authority_) external onlyOwner {
99       require(authority_ != address(0), "Authority address provided invalid");
100      _authority = authority_;
101  }
102
103  function setAggregator(address aggregatorProxyAddress_) external onlyOwner {
104      require(aggregatorProxyAddress_ != address(0), "Aggregator address provided invalid");
105      _aggregatorProxyAddress = aggregatorProxyAddress_;
106  }
```

Figure 12 Source code of NodeRewardVault contract (Unfixed)

```
114  function setBaseURI(string calldata baseURI) external onlyOwner {
115    _baseTokenURI = baseURI;
116  }
117
118  function withdrawMoney() external nonReentrant onlyOwner {
119    payable(owner()).transfer(address(this).balance);
120  }
121
122  function setAggregator(address aggregatorProxyAddress_) external onlyOwner {
123    _aggregatorProxyAddress = aggregatorProxyAddress_;
124    aggregator = IAggregator(_aggregatorProxyAddress);
125  }
```

Figure 13 Source code of ValidatorNft contract (Unfixed)

```
187  function setIsOpenSeaProxyActive(bool isOpenSeaProxyActive_)
188      external
189      onlyOwner
190  {
191      _isOpenSeaProxyActive = isOpenSeaProxyActive_;
192  }
```

Figure 14 Source code of *setIsOpenSeaProxyActive* function (Unfixed)

Recommendations	It is recommended to declare and trigger the corresponding event.
Status	Fixed. The project party has declared and triggered the corresponding event.

```
function transfer(uint256 amount, address to) external override nonReentrant onlyAggregator {
    require(to != address(0), "Recipient address provided invalid");
    payable(to).transfer(amount);
    emit Transferred(to, amount);
}

function setComission(uint256 comission_) external onlyOwner {
    require(comission_ < 10000, "Comission cannot be 100%");
    emit ComissionChanged(_comission, comission_);
    _comission = comission_;
}

function setTax(uint256 tax_) external onlyOwner {
    require(tax_ < 10000, "Tax cannot be 100%");
    emit TaxChanged(_tax, tax_);
    _tax = tax_;
}

function setDao(address dao_) external onlyOwner {
    require(dao_ != address(0), "DAO address provided invalid");
    emit DaoChanged(_dao, dao_);
    _dao = dao_;
}

function setAuthority(address authority_) external onlyOwner {
    require(authority_ != address(0), "Authority address provided invalid");
    emit AuthorityChanged(_authority, authority_);
    _authority = authority_;
}

function setAggregator(address aggregatorProxyAddress_) external onlyOwner {
    require(aggregatorProxyAddress_ != address(0), "Aggregator address provided invalid");
    emit AggregatorChanged(_aggregatorProxyAddress, aggregatorProxyAddress_);
    _aggregatorProxyAddress = aggregatorProxyAddress_;
}
```

Figure 15 Source code of NodeRewardVault contract (Fixed)

```
function setBaseURI(string calldata baseURI) external onlyOwner {
  emit BaseURIChanged(_baseTokenURI, baseURI);
  _baseTokenURI = baseURI;
}

function withdrawMoney() external nonReentrant onlyOwner {
  emit Transferred(owner(), address(this).balance);
  payable(owner()).transfer(address(this).balance);
}

function setAggregator(address aggregatorProxyAddress_) external onlyOwner {
  require(aggregatorProxyAddress_ != address(0), "Aggregator address provided invalid");
  emit AggregatorChanged(_aggregatorProxyAddress, aggregatorProxyAddress_);
  _aggregatorProxyAddress = aggregatorProxyAddress_;
  aggregator = IAggregator(_aggregatorProxyAddress);
}
```

Figure 16 Source code of ValidatorNft contract (Fixed)

```
function setIsOpenSeaProxyActive(bool isOpenSeaProxyActive_)
    external
    onlyOwner
{
  emit OpenSeaState(isOpenSeaProxyActive_);
  _isOpenSeaProxyActive = isOpenSeaProxyActive_;
}
```

Figure 17 Source code of *setIsOpenSeaProxyActive* 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.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com