BEOSIN
Blockchain Security

RiseCoin

Smart Contract Security Audit

No. 202505211437

May 21st, 2025

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 Low risk and 3 Info items were identified in the RiseCoin project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Project Description:

1. Basic Token Information

Token name	RiseCoin
Token symbol	RiseCoin
Token decimals	18
Total Supply	1,000,000,000(Burnable)
Token type	BEP-20

Table 1 RiseCoin token info

2. Business overview

RiseCoin is an BEP-20 token contract on the BNB Chain with a total supply of 1 billion tokens (1_000_000_000 * 10^18). Its core design includes: when `maxWalletEnable` is enabled, the maximum holding per address is 1 million tokens (`maxWallet`); the maximum sell ratio (`maxSellRate`) is set to 1% (100/10000), limiting single transaction sell amounts; a 10-second transaction cooldown (`cooldownSec`) prevents high-frequency trading. Buy and sell fees are set by `buyFundFee` and `sellFundFee`, both at 5% (500/10000). The `tradingEnable` flag controls trading activation, and `liquidityAdded` indicates whether liquidity has been added to prevent duplicate operations. The token burning mechanism, controlled by `isBurn`, activates when the contract's token balance exceeds 30 million tokens (`burnStartAmount`), targeting a reduction to 21 million tokens (`burnEndAmount`), with a burn rate (`burnRate`) of 88% (8800/10000), alongside a 2% service fee (`serviceRate`) and a 10% node fee (`nodeRate`). During sell transactions, `bnbRate` (45%) and `tokenRate` (55%) determine the allocation between ETH and tokens. Priority trading windows are set via `nodePriorityTime` (600 seconds) and `whitePriorityTime` (300 seconds), working with `nodeList` and `whiteList` to manage transaction limits for nodes and whitelisted addresses, respectively. The `tradingOpenTime` specifies the market opening time for flexible control. The `swapAddress` marks decentralized exchange addresses, with associated buy and sell fees set by `swapBuyFee` and `swapSellFee`, both at 5%. The `excludeCoolingOf` mapping records addresses exempt from cooldowns and fees.

1 Overview

1.1 Project Overview

Project Name	RiseCoin
Project Language	Solidity
Platform	BNB Chain
Sha256	e465d63e95f3a540dc4b9ec4d60ffef1a6929154429b13ffd5ea73e678727f86 8d45e9e280032ca94045541263595b82dd0704f81b0c1aa2dbdb9eadb3ffa03c

1.2 Audit Overview

Audit work duration: May 20, 2025 - May 21, 2025

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
RiseCoin-01	Privilege management risk	Low	Fixed
RiseCoin-02	Parameter Range Unlimited	Info	Fixed
RiseCoin-03	Redundant codes	Info	Fixed
RiseCoin-04	Lack of event triggering	Info	Fixed

Finding Details:

[RiseCoin-01] Privilege management risk

Severity Level	Low
Type	Business Security
Lines	RiseToken.sol #L48-58
Description	The current contract uses a multi-mapping type `_administors`, allowing multiple administrators to be set, which is not recommended as it may lead to the abuse of administrative privileges.

```
    function setAdministor(
        address newAdmin,
        bool _status
    ) public virtual onlyOwner {
        _setAdminship(newAdmin, _status);
    }

    function _setAdminship(address newAdmin, bool _status) internal
virtual {
        _administors[newAdmin] = _status;
        emit AdminhipTransferred(newAdmin, _status);
    }
```

Recommendation	It is recommended to switch to a single administrator address or implement a stricter access control mechanism to enhance security.
Status	**Fixed.** The project team has been modified to have a single administrator to manage permissions.

```
    function setAdministor(address newAdmin) public virtual onlyOwner
{
        _setAdminship(newAdmin);
    }
    function _setAdminship(address newAdmin) internal virtual {
        address oldAdmin = _administors;
        _administors = newAdmin;
        emit AdminhipTransferred(oldAdmin, newAdmin);
    }
```

[RiseCoin-02] Parameter Range Unlimited

Severity Level	Info
Type	Business Security
Lines	RiseToken.sol #L446-459,467-476
Description	In the `setSwapAddress` and `setBurnParam` functions, when the administrator modifies critical parameters (such as swap address status, buy/sell fee rates, burn activation, burn start amount, burn end amount, burn rate, service rate, and node rate), there is no restriction on the parameter values' reasonable range, which could lead to user losses due to excessively large values.

```
function setBurnParam(
    bool isb,
    uint256 bst,
    uint256 bet,
    uint256 brate,
    uint256 srate,
    uint256 nrate
) external onlyAdministor {
    isBurn = isb;
    burnStartAmount = bst;
    burnEndAmount = bet;
    burnRate = brate;
    serviceRate = srate;
    nodeRate = nrate;
}
function setSwapAddress(
    address _swap,
    bool _status,
    uint256 _buyfee,
    uint256 _sellfee
) external onlyAdministor {
    swapAddress[_swap] = _status;
    swapBuyFee = _buyfee;
    swapSellFee = _sellfee;
}
```

Recommendation	It is recommended that parameter range validation logic be added to ensure that all parameters are within safe and reasonable ranges.

Status **Fixed.** The project team has added parameter range validation logic.

```
    function setSwapAddress(
        address _swap,
        bool _status,
        uint256 _buyfee,
        uint256 _sellfee
    ) external onlyAdministor {
        require(swapBuyFee <= 10000, "rate is over 10000");
        require(swapSellFee <= 10000, "rate is over 10000");
        swapAddress[_swap] = _status;
        swapBuyFee = _buyfee;
        swapSellFee = _sellfee;
        emit SwapAddressLog(_msgSender(), _swap, _status, _buyfee,
_sellfee);
    }
    function setBurnParam(
        bool isb,
        uint256 bst,
        uint256 bet,
        uint256 brate,
        uint256 srate,
        uint256 nrate
    ) external onlyAdministor {
        require(
            burnStartAmount > burnEndAmount,
            "start amount must bigger than end amount"
        );
        require(burnEndAmount > 10_000_000 * 10 ** 18, "error end
amount");
        require(
            burnRate + serviceRate + nodeRate <= 10000,
            "rate is over 10000"
        );
    ......
    }
```

[RiseCoin-03] Redundant codes

Severity Level	Info
Type	Coding Conventions
Lines	RiseToken.sol #L433-439,786
Description	The `isContract` function in the RiseCoin contract is redundant code and is not called in the contract. In RiseCoin contracts, the upper logic of the sell function already checks for `tradingEnable` status, so it is redundant to double-check `tradingEnable` inside the `sell` function.
Recommendation	It is recommended to remove the `isContract` function from RiseCoin contracts and remove the `tradingEnable` status check from the `sell` function.
Status	**Fixed.** The project team has removed the redundant code.

```
function isContract(address _addr) private view returns (bool) {
    uint32 size;
    assembly {
        size := extcodesize(_addr)
    }
    return (size > 0);
  }

function sell(uint256 sell_amount) internal {

    require(tradingEnable, "Trading not enable");
```

[RiseCoin-04] Lack of event triggering

Severity Level	Info
Type	Coding Conventions
Lines	RiseToken.sol #L478-498
Description	In RiseCoin contracts, when administrators modify key parameters (such as the account's cooling period exemption status and node allocation amount) through the `setExcludeCoolingOf` and `setNodes` functions, no corresponding events are triggered, so it is recommended that an event triggering mechanism be added to improve transparency and traceability.
Recommendation	It is recommended to trigger events when important parameters are modified.
Status	**Fixed.** The project team has added event triggers.

```
    function setExcludeCoolingOf(
        address[] memory accounts,
        bool _ok
    ) external onlyAdministor {
        for (uint256 i = 0; i < accounts.length; i++) {
            excludeCoolingOf[accounts[i]] = _ok;
        }
    }
    function setNodes(
        address[] calldata accounts,
        uint256[] calldata amounts
    ) external onlyAdministor {
        require(accounts.length == amounts.length, "dismatch length");
        for (uint256 i = 0; i < accounts.length; i++) {
            address to = accounts[i];
            uint256 amount = amounts[i];
            nodeList[to] = amount;
        }
    }
```

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

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

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

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

3.1.2 Degree of impact

- **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.

3.1.3 Likelihood of Exploitation

- **Probable**

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

- **Possible**

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

- **Unlikely**

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

- **Rare**

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

3.1.4 Fix Results Status

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

3.2 Audit Categories

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

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

- **Coding Conventions**

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

- **General Vulnerability**

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

- **Business Security**

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

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

3.3 Disclaimer

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

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

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

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

3.4 About Beosin

Beosin is 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