BEOSIN
Blockchain Security

RunesRouter

Smart Contract Security Audit

No. 202405101428

May 10th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

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

High	Fixed: 2	Acknowledged: 0
Low	Fixed: 2	Acknowledged: 0
Info	Fixed: 1	Acknowledged: 0

- **Project Description:**

Business overview

RunesRouter is a routing contract for cross-chain bridges. It primarily consists of four main functionalities:The first one is the management of a whitelist for cross-chain tokens. The owner can call the `addToken` and `removeToken` functions to add or remove tokens that are supported by the cross-chain bridge.The second functionality is the management of validators. The owner can call the `addValidator` and `removeValidator` functions to add or remove validators.The third functionality is the deposit of tokens that need to be transferred across chains. Users can call the `deposit` function to deposit tokens that require cross-chain transfers into the contract.The fourth functionality is the withdrawal of cross-chain tokens. Users can call the `withdraw` function and use the off-chain validator's signature to withdraw funds on the target chain.

1 Overview

1.1 Project Overview

Project Name	RunesRouter
Project Language	Solidity
Platform	ZetaChain,Ethereum
Code Base	https://github.com/runesbridge/runesbridge-contracts/blob/main/src/RunesRouter.sol
commit	eefc2a6a5e80ed8a2e4cdb45f01a89563ec806d7 9ad2047d627deb0dfb5634a99e0533524d0b6749

1.2 Audit Overview

Audit work duration: May 09, 2024 – May 10, 2024

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
RunesRouter-01	The chainId is not being validated	High	Fixed
RunesRouter-02	Lack of the limitation of Validators	High	Fixed
RunesRouter-03	The cross-chain amount calculates error	Low	Fixed
RunesRouter-04	Signature Reuse Risk	Low	Fixed
RunesRouter-05	Missing Event Trigger	Info	Fixed

Finding Details:

[RunesRouter-01] The chainId is not being validated

Severity Level	High
Type	Business Security
Lines	RunesRouter.sol #L102-148
Description	Users can call the `withdraw` function and extract cross-chain funds. However, the contract does not verify whether the chainID in the signature is consistent with the actual cross-chain `chainID`. Therefore, in the case where the token and `validators` addresses are the same, an attacker can execute a withdrawal transaction on Chain A, which can also be executed on Chain B.

```
function withdraw(
    address token,
    string memory from,
    uint256 amount,
    string memory txhash,
    uint256 chainId,
    bytes[] calldata signatures
) external whenNotPaused nonReentrant notProcessed(txhash) {
    require(
        signatures.length == _validators.length,
        "invalid length of signatures"
    );
    for (uint i = 0; i < _validators.length; i++) {
        require(
            _verify(
                token,
                from,
                _msgSender(),
                amount,
                txhash,
                signatures[i],
                chainId,
                _validators[i]
            ),
            "invalid signature"
```

	``); } txProcessed[txhash] = true; IERC20(token).transfer(_msgSender(), amount); emit Withdraw(token, from, _msgSender(), amount, txhash, chainId); } ```
Recommendation	It is recommended to check whether the chain ID for the withdraw is the same as the local chain ID.
Status	**Fixed.** The project team modified the relevant code to check chain ID.

[RunesRouter-02] Lack of the limitation of Validators

Severity Level	High
Type	Business Security
Lines	RunesRouter.sol #L84-100
Description	There is no limitation on the number of Validators that can be removed through the `removeValidator` function in the contract. If the owner removes all Validators from the contract, anyone, including the owner, would be able to withdraw all the tokens from the contract.

```
function removeValidator(address _address) external onlyOwner {
    require(isValidator[_address], "address not exist");
    require(indexes[_address] < _validators.length, "index out of range");
    uint256 index = indexes[_address];
    uint256 lastIndex = _validators.length - 1;
    if (index != lastIndex) {
        address lastAddr = _validators[lastIndex];
        _validators[index] = lastAddr;
        indexes[lastAddr] = index;
    }
    delete isValidator[_address];
    delete indexes[_address];
    _validators.pop();
}
```

Recommendation	It is recommended to add the limitation of the Validators when owner remove.
Status	**Fixed.**The project team add a restriction has been added to the `removeValidator` function, specifying that validators can only be removed when their quantity is greater than 1.Furthermore, a validation check has been implemented in the `withdraw` function. Withdrawals are only allowed when validators exist.

[RunesRouter-03] The cross-chain amount calculates error

Severity Level	Low
Type	Business Security
Lines	RunesRouter.sol #L102-115
Description	In best practices not to use the change in the sender's token balance as cross-chain funds.If the deposited token is a deflationary token, there may be a certain amount of fees charged to the user during the transfer. This fee should not be included in the cross-chain amount. Therefore, if the difference in the sender's balance is used as the actual cross-chain amount for the user, it would result in the cross-chain bridge contract losing funds.

```
function deposit(
    address token,
    string memory to,
    uint256 amount,
    uint256 chainId
) external whenNotPaused {
    require(acceptedTokens[token], "token not accepted");
    uint256 balance = IERC20(token).balanceOf(_msgSender());
    IERC20(token).transferFrom(_msgSender(), address(this), amount);
    uint256 newBalance = IERC20(token).balanceOf(_msgSender());
    amount = balance - newBalance;
    emit Deposit(token, _msgSender(), to, amount, chainId);
}
```

Recommendation	It is recommended to use the difference in token balance of the contract itself as the actual cross-chain amount for the user.
Status	**Fixed.**The project team replaces the original sender's balance change with the received token amount at the contract as the cross-chain amount.

[RunesRouter-04] Signature Reuse Risk

Severity Level	Low
Type	Business Security
Lines	RunesRouter.sol #L54-63
Description	In the initialize function of the contract, the `__EIP712_init` function of the `EIP712Upgradeable` contract is not called to initialize the parameters `name` and `version`. These parameter is designed to include bits of project unique information such as the name of the project. If these parameters are not initialized, it may allow other contracts with the same signature structure as this contract to pass verification. For example, if there are two different versions of the Router contract with identical signature structures after this contract is upgraded, the original withdrawal transactions can be successfully executed in both versions of the Router.

```
function initialize(
    address _validator1,
    address _validator2,
    address _validator3
) public initializer {
    _addValidator(_validator1);
    _addValidator(_validator2);
    _addValidator(_validator3);
    __Ownable_init(_msgSender());
}
```

Recommendation	It is recommended to use `__EIP712_init` function when initialize.
Status	**Fixed.**

```
function initialize(
    address _validator1,
    address _validator2,
    address _validator3
) public initializer {
    _addValidator(_validator1);
    _addValidator(_validator2);
    _addValidator(_validator3);
    __Ownable_init(_msgSender());
    __EIP712_init("RunesRouter", "1");
```

```
}
```

[RunesRouter-05] Missing Event Trigger

Severity Level	Info
Type	Coding Conventions
Lines	RunesRouter.sol#L108-135
Description	The Validator in the contract does not trigger events when key parameters are modified.

```
function addToken(address token) external onlyOwner {
    acceptedTokens[token] = true;

}

function removeToken(address token) external onlyOwner {

    acceptedTokens[token] = false;

}

function addValidator(address _address) public onlyOwner {

    _addValidator(_address);

}

function _addValidator(address _address) internal {

    require(!isValidator[_address], "already exist");

    indexes[_address] = _validators.length;

    isValidator[_address] = true;

    _validators.push(_address);

}

function removeValidator(address _address) external onlyOwner {

    require(isValidator[_address], "address not exist");

    require(indexes[_address] < _validators.length, "index out of
range");

    uint256 index = indexes[_address];

    uint256 lastIndex = _validators.length - 1;
```

	`` if (index != lastIndex) { address lastAddr = _validators[lastIndex]; _validators[index] = lastAddr; indexes[lastAddr] = index; } delete isValidator[_address]; delete indexes[_address]; _validators.pop(); } ```
Recommendation	It is recommended to modifying critical variables is a recommended practice as it provides a standardized way to capture and communicate important changes within the contract. Events enable transparency and allow external systems and users to easily track and react to these modifications.
Status	**Fixed.**The project team added the corresponding event.

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	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.

3.1.5 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/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com