BEOSIN
Web3 Security & Compliance

RateX-Protocol

Smart Contract Security Audit

No. 202512091642

Dec 9th, 2025

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

1 Overview **5**

1.1 Project Overview 5

1.2 Audit Overview 5

1.3 Audit Method 5

2 Findings **7**

[RateX-Protocol-01] fund_vault lacks state binding validation 8

[RateX-Protocol-02] Missing emergency withdraw function 9

[RateX-Protocol-03] Redundant code 10

[RateX-Protocol-04] Missing events 11

3 Appendix **12**

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 12

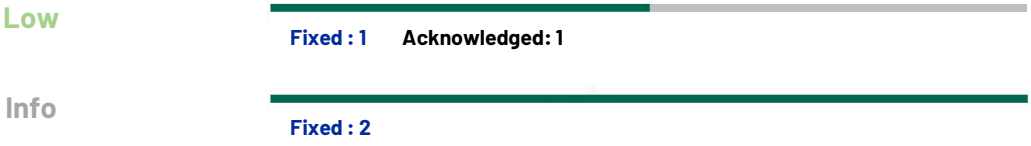
3.2 Audit Categories 15

3.3 Disclaimer 17

3.4 About Beosin 18

Summary of Audit Results

After auditing, 2 Low and 2 info risk items were identified in the RateX-Protocol 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. Business overview

The audited RateX-Protocol project consists of two main modules: airdrop and vesting.

- The Airdrop contract is a multi-stage, upgradeable token distribution and linear vesting system. The project team assigns each user's total claimable token amount off-chain via signed messages. Users claim their tokens using claimWithSignature — upon the first successful claim, the EIP-712 signature is verified, the user's total entitlement is recorded, and the initial TGE (Token Generation Event) portion is immediately released. Subsequent claims are made via a simple claim() call; as each predefined stage timestamp is reached, the corresponding additional tokens are automatically unlocked and transferable.

  Stage schedules are configured with precise basis points (bps) and Unix timestamps, while per-user claimed tracking prevents double-claiming. The contract includes reentrancy guards, a pause mechanism, a changeable signer address, emergency withdrawal for the owner, and follows the UUPS proxy pattern for secure future upgrades.

- The Vesting contract is a flexible, multi-plan token escrow and linear release system. Project teams can create multiple independent vesting plans via createPlan(), defining key parameters such as start timestamp, cliff duration, cliff unlock ratio (in bps), linear vesting duration, and release frequency. Once a plan is created, the owner registers beneficiaries and their respective token allocations, then funds each user's dedicated escrow vault in one or more batches.

  Users can call claim() at any time after the cliff or during/after the linear period to automatically receive the currently unlocked portion. Release schedules are driven by consistent bps and timestamp logic, with individual claimed amounts tracked per user to prevent duplicate withdrawals. The design ensures transparent, predictable, and fully automated token vesting for various stakeholder groups.

1 Overview

1.1 Project Overview

Project Name	RateX-Protocol
Project Language	Solidity, Rust
Platform	BNB Chain, Solana
Github link	https://github.com/RateX-Protocol/ratex-token-airdrop https://github.com/RateX-Protocol/ratex-token-vesting
Commit	d38dedaf1bd9b64850e32972a34d45e893def8d5 (airdrop initial) 6ef166d61bbfe2a760218d39f25f40c4f5683feb (airdrop latest) 3381505b1d43011504fcc28c31335106e5b5eb94 (vesting initial) 2752e118957aefce52371bced6a41ed599389034 (vesting latest)

1.2 Audit Overview

Audit work duration:Nov 21 - Nov 24, Dec 9, 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
RateX-Protocol-01	fund_vault lacks state binding validation	Low	Acknowledged
RateX-Protocol-02	Missing emergency withdraw function	Low	Fixed
RateX-Protocol-03	Redundant code	Info	Fixed
RateX-Protocol-04	Missing events	Info	Fixed

Finding Details:

[RateX-Protocol-01] fund_vault lacks state binding validation

Severity Level	Low
Type	Business Security
Lines	lib.rs#L179-204

Description	The `fund_vault` instruction relies solely on caller-provided accounts and performs only minimal consistency checks. This allows any caller to mark a properly structured (but unapproved) vesting account as funded, bypassing the intended admin-governed workflow and compromising process integrity, even though no direct token loss occurs.
	`pub fn fund_vault(ctx: Context<FundVault>) -> Result<()> {`
	`// Check if already funded`
	`require!(!ctx.accounts.user_vesting.is_funded, VestingError::AlreadyFunded);`
	`// Ensure mint consistency`
	`require!(ctx.accounts.user_vesting.mint == ctx.accounts.mint.key(), VestingError::MintMismatch);`

Recommendation	It is recommended to validate the provided `user_vesting` account against on-chain configuration and plan authority rather than trusting input accounts.
Status	**Acknowledged.**

[RateX-Protocol-02] Missing emergency withdraw function

Severity Level	Low
Type	Business Security
Lines	TokenAirdrop.sol
Description	The contract does not provide an owner-only withdrawal function for remaining tokens in the airdrop vault. If some users fail to claim their allocation before or after the vesting period ends, the unclaimed tokens become permanently locked, resulting in unnecessary capital lockup.
Recommendation	It is recommended to implement an owner-only extraction function that allows the admin to withdraw any remaining tokens after a reasonable deadline or when claims are no longer expected.
Status	**Fixed.**

[RateX-Protocol-03] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	lib.rs#L92
Description	The `AlreadyInitialized` error variant is defined but never used. `AlreadyInitialized,`
Recommendation	It is recommended to remove the unused `AlreadyInitialized` error variant.
Status	**Fixed.**

[RateX-Protocol-04] Missing events

Severity Level	Info
Type	Coding Conventions
Lines	TokenAirdrop.sol#L241-250
Description	Functions that modify important parameters do not emit events, reducing transparency and hindering off-chain monitoring. `function updateSigner(address newSigner) external onlyOwner { if (newSigner == address(0)) revert ZeroAddress(); if (newSigner == signer) revert SameValue(); signer = newSigner; } function setPaused(bool _paused) external onlyOwner { if (_paused == paused) revert SameValue(); paused = _paused; }`
Recommendation	It is recommended to add corresponding events in `updateSigner` and `setPaused`.
Status	**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.

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

- **Critical**

Critical 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 Critical 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	Deprecated Items
		Redundant Code
		require/assert Usage
		Default Values
2	General Vulnerability	Insufficient Address Validation
		Lack Of Address Normalization
		Variable Override
		DoS (Denial Of Service)
		Function Call Permissions
		Call/Delegatecall Security
		Tx.origin Usage
		Returned Value Security
		Mathematical Risk
		Overriding Variables
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Arbitrage Attack
		Access Control

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 Rust 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.

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 a leading blockchain security and compliance technology company established in 2018. Being focused on blockchain ecosystem security and compliance, it has developed a product matrix including Beosin KYT, Beosin Trace, and Stablecoin Monitor, which have obtained international certifications such as ISO 27001 and SOC 2. Beosin's core products have been applied for over 70 intellectual property rights, and the company has participated in the development of multiple international standards related to blockchain security. It was among the first batch of enterprises selected for the Cyberport Incubation Programme. Its business covers professional code security audit services for blockchain ecosystems, anti-money laundering compliance technology services for exchanges, financial institutions, and payment institutions, and virtual asset tracing and investigation services for law enforcement and regulatory authorities.

As one of the earliest companies to apply formal verification to blockchain security, Beosin offers professional blockchain and smart contract security audit services. Beosin has audited over 4,500 smart contracts and blockchain projects and has become the official security partner for several renowned blockchains, including BNB Chain, TON, Soneium, Manta Network, Sonic SVM, and SOON Network.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin/