BEOSIN
Web3 Security & Compliance

RateX

Smart Contract Security Audit

No. 202505161749

May 16th, 2025

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 4 info items were identified in the RateX project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Info	
	Fixed : 4 Acknowledged: 0

- **Project Description:**

The RateX project is built around five core functions: adding/withdrawing margin, trading, adding/withdrawing liquidity, clearing accounts and collecting earnings. The features are described in detail below:

✧ **Add and withdraw margins:**

The `handle_deposit` function allows users to deposit tokens into a margin market. It checks the market's status, updates the user's margin position and the market's balance, transfers tokens from the user's wallet to the market vault, and logs all details with a `DepositRecord`. The `handle_multi_sig_deposit` function is similar but adds extra security checks for multi-signature accounts, ensuring every step is verified and recorded for safety. Conversely, the `handle_withdraw` function precisely handles withdrawals. It retrieves user details, confirms the market's status, and ensures the vault has enough funds to fulfill the withdrawal request. The function deducts the withdrawal amount, transfers tokens back to the user's account, and records a `DepositRecord` with all the details. It also performs checks to ensure the user's remaining positions meet collateral requirements, keeping everything compliant.

✧ **Trade:**

The `handle_place_order` function is the starting point for trading. Users create orders in the yield market, which are later executed by keepers through `fill_order`. It retrieves the current time and user information, charges a small keeper fee to keep the system running, and loads market data. After confirming the market is active, it cancels any expired orders, verifies the new order's details (such as size and expiration), and generates a unique order ID. The order is saved, logged with an `OrderRecord`, and set to "open" status, awaiting keeper matching and settlement. This setup ensures orders are securely placed and ready for subsequent execution.

✧ **Adding and Removing Liquidity:**

The `handle_add_lp_shares` function enables users to add liquidity to the yield market, converting their tokens into a share of the trading pool. It verifies the deposit amount, market status, and rate boundaries, then calculates the liquidity provided by the user's tokens. Tokens are transferred, positions are updated, and both a `DepositRecord` and `LPRecord` are logged to track the operation. In contrast, `handle_remove_lp_shares` allows users to withdraw their liquidity share. It validates the withdrawal percentage, handles any social losses (such as market imbalances), and swaps assets back to the user. The function updates market and user states, logs the withdrawal, and ensures compliance with market rules.

✧ **Investing and Redeeming Yields:**

The handle_earn_invest function lets users invest margin in the yield market, converting it into quote assets and allocating them to a yield position. It charges fees, mints PT tokens to represent the investment share, and logs all details via an EarnRecord. Conversely, handle_earn_redeem enables users to redeem assets by burning PT tokens. It calculates the redeemable amount based on the redemption ratio, swaps assets back, deducts fees, and returns margin to the user, logging another EarnRecord. Both functions keep market and vault states synchronized, ensuring seamless and secure investment and redemption.

✧ **Liquidation:**

The handle_liquidate, handle_liquidate_lp, and handle_liquidate_insurance functions address accounts that fail to meet collateral requirements. handle_liquidate targets user accounts, transferring undercollateralized yield positions to an insurance account while charging fees and logging transfer details. handle_liquidate_lp clears liquidity provider positions, removing liquidity and transferring assets to the insurance account, updating market counts, and logging the process. handle_liquidate_insurance manages the insurance account itself, offsetting positions through automated deleveraging (ADL) or transferring losses to a social loss pool, with detailed records of each step. These functions maintain market stability by resolving risky positions.

✧ **Claiming Yields:**

The handle_claim_yield function allows users in spot markets to claim accumulated yields. It examines the user's yield positions, recalculates earned yields using oracle data, converts them to margin assets, and withdraws the requested amount to the user's wallet. The function ensures the market is active, verifies collateral requirements post-withdrawal, and logs transaction details via a ClaimYieldRecord, including timestamp, user authority, and withdrawal amount. This process ensures users can reliably and securely access their yields.

1 Overview

1.1 Project Overview

Project Name	RateX
Project Language	Rust
Platform	Solana
Github Link	https://github.com/RateX-Protocol/ratex-program/tree/dev-alpha-anchor-0.30.1
Commit	324bf66d4e533122bb575809db2bf955a409097b f79812a366a804ecf6b8122f0ac87207cf7b0bed

1.2 Audit Overview

Audit work duration: Apr 27, 2024 – May 16, 2025

Update time: May 20, 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-01	Multi-Load	Info	Fixed
RateX-02	Redundant codes	Info	Fixed
RateX-03	Poorly designed update_protocol_fee function	Info	Fixed
RateX-04	Branch redundancy	Info	Fixed

Finding Details:

[RateX-01] Multi-Load

Severity Level	Info
Lines	src\instructions\liquidation_insurance.rs#24-262
Type	Coding Conventions
Description	In the `handle_liquidate_insurance` function, ctx.accounts.yield_market is loaded multiple times (via load() or load_mut()), and this repeated loading increases performance overhead on high throughput blockchains such as Solana due to the high cost of the account loading operation. Similar problems exist with other functions, which will not be described subsequently.
Recommendation	It is recommended to optimize the number of loads, try to load yield_market once at the beginning of the function and pass references when needed.
Status	**Fixed.**

[RateX-02] Redundant codes

Severity Level	Info
Type	Coding Conventions
Description	In the `liquidate_position_adl` function of controller\liquidation.rs, the parameters `order_step_size` and `min_liquidation_size` are defined but not used; in the `handle_liquidate_insurance` function, the parameter `base_asset_amount` is defined but not used. In the `handle_liquidate_insurance` function, the `base_asset_amount` parameter is defined but not used, which is redundant code. `market_map`, `_oracle_map`, and `_slot` are not used.
Recommendation	It is recommended that code redundancy be removed to improve readability and maintainability and to reduce the potential overhead of execution in the Solana chain.
Status	**Fixed.**

[RateX-03] Poorly designed update_protocol_fee function

Severity Level	Info
Lines	src\controller\earn.rs#159-161
Type	Business Security
Description	In the `update_protocol_fee` function, zero can be returned directly after the `earn_vault.yield_position.last_rate == 0` branch to avoid unnecessary gas consumption.
Recommendation	It is recommended to return zero directly after executing the `earn_vault.yield_position.last_rate == 0` branch.
Status	**Fixed.**

```
pub fn update_protocol_fee(earn_vault: &mut EarnVault, oracle:
&Oracle) -> Result<(i64)> {
    let user_ratio: u64 = earn_vault.user_ratio;
    let mut cumulative_yield_from_realized: i64 = 0;
    if earn_vault.yield_position.last_rate == 0 {
        earn_vault.yield_position.last_rate = oracle.rate;
    }
```

[RateX-04] Branch redundancy

Severity Level	Info
Lines	src\controller\earn.rs#45-50,108-113
Type	Business Security
Description	In the `earn_invest` function, `other_amount_threshold` is always 0 because `a_to_b` is always true, and the true branch of match `!a_to_b` (u64::MAX) is never executed, making the code redundant. In the `earn_redeem` function, the false branch (0) of match `!a_to_b` is never executed, again redundant.
Recommendation	It is recommended to remove redundant branches to improve code readability.
Status	**Fixed.**

```
let a_to_b = true;
let amount_specified_is_input = a_to_b;
let other_amount_threshold = match !a_to_b {
    true => u64::MAX,
    false => 0,
};
let a_to_b = false;
let amount_specified_is_input = a_to_b;
let other_amount_threshold = match !a_to_b {
    true => u64::MAX,
    false => 0,
};
```

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 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	SPL Token Standards
		Visibility Specifiers
		Lamport Check
		Account Check
		Signer Check
		Program Id Check
		Deprecated Items
		Redundant Code
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		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
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/