OFFSIDE LABS

RateX

Smart Contract Security Assessment

June 2025

Prepared for:

Rate Labs Limited

Prepared by:

Offside Labs

Yao Li

Ronny Xing

Contents

1 About Offside Labs

Offside Labs stands as a pre-eminent security research team, comprising highly skilled hackers with top - tier talent from both academia and industry.

The team demonstrates extensive and diverse expertise in modern software systems, which encompasses yet are not restricted to *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. Offside Labs is at the forefront of innovative domains such as *cryptocurrencies* and *blockchain technologies*. The team achieved notable accomplishments including the successful execution of remote jailbreaks on devices like the **iPhone** and **PlayStation 4**, as well as the identification and resolution of critical vulnerabilities within the **Tron Network**.

Offside Labs actively involves in and keeps contributing to the security community. The team was the winner and co-organizer for the *DEFCON CTF*, the most renowned CTF competition in Web2. The team also triumphed in the **Paradigm CTF 2023** in Web3. Meanwhile, the team has been conducting responsible disclosure of numerous vulnerabilities to leading technology companies, including *Apple*, *Google*, and *Microsoft*, safeguarding digital assets with an estimated value exceeding **$300 million**.

During the transition to Web3, Offside Labs has attained remarkable success. The team has earned over **$9 million** in bug bounties, and **three** of its innovative techniques were acknowledged as being among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

2 Executive Summary

Introduction

Offside Labs completed a security audit of *RateX* smart contracts, starting on May 14, 2025, and concluding on June 17, 2025.

Project Overview

RateX is a cross-chain margin and spot yield trading protocol that enables users to trade Yield Tokens of various yield-bearing assets with leverage, allowing for capital-efficient participation in yield movements. In addition to leveraged yield trading, RateX offers two key features: Earn Fixed Yield, which allows users to lock in stable returns, and Yield Liquidity Farming, which enables users to enhance their earnings by providing liquidity. Through these functionalities, RateX caters to diverse user needs, combining yield tokenization, yield and principal splitting, and trading into a seamless platform.

Audit Scope

The assessment scope contains mainly the smart contracts of the RateX program for the *RateX* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- RateX
  - Codebase: `https://github.com/RateX-Protocol/ratex-program`
  - Commit Hash:
    - 324bf66d4e533122bb575809db2bf955a409097b

We listed the files we have audited below:

- RateX
  - programs/ratex-contracts/src/controller/deposit.rs
  - programs/ratex-contracts/src/controller/earn.rs
  - programs/ratex-contracts/src/controller/epoch_update.rs
  - programs/ratex-contracts/src/controller/keeper_fee.rs
  - programs/ratex-contracts/src/controller/liquidation.rs
  - programs/ratex-contracts/src/controller/orders.rs
  - programs/ratex-contracts/src/controller/position.rs
  - programs/ratex-contracts/src/controller/swap_manager.rs
  - programs/ratex-contracts/src/controller/token.rs
  - programs/ratex-contracts/src/controller/withdraw.rs
  - programs/ratex-contracts/src/instructions/admin.rs
  - programs/ratex-contracts/src/instructions/amm.rs
  - programs/ratex-contracts/src/instructions/earn.rs
  - programs/ratex-contracts/src/instructions/liquidation.rs

- programs/ratex-contracts/src/instructions/liquidation_insurance.rs
- programs/ratex-contracts/src/instructions/lp.rs
- programs/ratex-contracts/src/instructions/optional_accounts.rs
- programs/ratex-contracts/src/instructions/swap.rs
- programs/ratex-contracts/src/instructions/user.rs
- programs/ratex-contracts/src/instructions/view.rs
- programs/ratex-contracts/src/math/amm/swap_math.rs
- programs/ratex-contracts/src/math/bn.rs
- programs/ratex-contracts/src/math/margin.rs
- programs/ratex-contracts/src/math/safe_math.rs
- programs/ratex-contracts/src/math/token.rs
- programs/ratex-contracts/src/math/yield_market.rs
- programs/ratex-contracts/src/state/amm.rs
- programs/ratex-contracts/src/state/earn.rs
- programs/ratex-contracts/src/state/epoch_update_account.rs
- programs/ratex-contracts/src/state/events.rs
- programs/ratex-contracts/src/state/liquidation.rs
- programs/ratex-contracts/src/state/lp.rs
- programs/ratex-contracts/src/state/margin_market_map.rs
- programs/ratex-contracts/src/state/observation_state_map.rs
- programs/ratex-contracts/src/state/oracle.rs
- programs/ratex-contracts/src/state/oracle_map.rs
- programs/ratex-contracts/src/state/state.rs
- programs/ratex-contracts/src/state/user.rs
- programs/ratex-contracts/src/state/yield_market.rs
- programs/ratex-contracts/src/state/yield_market_map.rs
- programs/ratex-contracts/src/util/swap_tick_sequence.rs
- programs/ratex-contracts/src/util/token.rs
- programs/ratex-contracts/src/util/user.rs

Findings

The security audit revealed:

- 0 critical issue
- 1 high issues
- 3 medium issues
- 4 low issues
- 2 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Incorrect Amount Conversion in Liquidation	High	Fixed
02	Incorrect Liquidation Fee Accounting for Yield Market	Medium	Fixed
03	Lack of Fee Validation in DeleteLp IX	Medium	Fixed
04	Lack of Slippage Protection for Price and PT Amount in Earn Invest	Medium	Fixed
05	Zero Division May Cause DoS in Liquidation	Low	Fixed
06	Residual Input Amount Return Can Panic end_vault_swap IX	Low	Fixed
07	Price on Tick Boundary May Cause Zero Liquidity for Unified Position Calculation	Low	Fixed
08	Precision Loss in Yield Position Fee Calculation	Low	Fixed
09	Lack of Validation for UserAuthority in get_user_and_token_account Function	Informational	Acknowledged
10	Lack of Validation for other_amount_threshold in end_vault_swap IX	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Incorrect Amount Conversion in Liquidation

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `handle_liquidate` function, the calculation of `transfer_amount` for each yield position is flawed. The relevant code snippet is as follows:

```
142 let asset_value = margin_asset_value.safe_add(position_asset_value)?;
143 let liability_value =
    ↪ margin_liability_value.safe_add(position_liability_value)?;
144
145 let target_value = mul_div_i64(liability_value, COLLATERAL_RATIO_MAINTENANCE,
    ↪ PERCENT_PRECISION)?.abs();
146 let transfer_value = target_value.safe_sub(asset_value)?.max(0);
147 let transfer_amount = yield_market.quote_to_margin(transfer_value, &oracle)?;
```

programs/ratex-contracts/src/instructions/liquidation.rs#L142-L147

The issue arises because `transfer_value` is derived from `asset_value` and `liability_value`, which are in scaled margin. However, the conversion using `quote_to_margin` does not appropriately account for this unit.

Impact

This incorrect conversion results in an inaccurate margin amount to transfer. Consequently, this can lead to:

- Improper margin transfer distributions across all yield markets.
- Increased risk of certain `insurance_yield_position` becoming bad debt.
- Potential social losses borne by liquidity providers.

Recommendation

Unscaled the `transfer_value` instead of using `quote_to_margin`.

Mitigation Review Log

Fixed in commit 76c8a6fec0dda5379aa3076c583f23ada43097fc.

4.2 Incorrect Liquidation Fee Accounting for Yield Market

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `handle_liquidate` function, the accounting for the liquidation fee is as follows:

```
189 if yield_market.insurance_margin_position.balance > 0 {
190     let margin_amount = yield_market.insurance_margin_position.balance;
191     ...
192     yield_market.insurance_margin_position.balance = 0;
193 }
194 yield_market.protocol_fee =
195 yield_market.protocol_fee.safe_add(liq_fee_in_margin)?;
```

programs/ratex-contracts/src/instructions/liquidation.rs#L189-L195

However, if `insurance_margin_position.balance` equals 0, the accounting for the liquidation fee will be skipped.

Impact

If `insurance_margin_position.balance` is 0, the `net_quote_amount` and `net_quote_amount_realized` of the yield market will become incorrect. This can affect subsequent state updates and the rebalancing of margin vaults, potentially leading to DoS issues during withdrawal processes.

Recommendation

Remove the `if` condition.

Mitigation Review Log

Fixed in commit 4f5579a9a01902195c608d4aed99007d28dee8f4.

4.3 Lack of Fee Validation in DeleteLp IX

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

In the `delete_lp` instruction, the following checks are performed to prevent the deletion of non-empty LP position:

```
76      validate!(
77          lp.last_liquidity == 0
78              && lp.reserve_quote_amount == 0
79              && lp.reserve_base_amount == 0,
80          ErrorCode::LpCantBeDeleted
81      )?;
```

programs/ratex-contracts/src/instructions/lp.rs#L76-L81

However, there is no validation to check if the fee field of the LP is zero.

Impact

The lack of zero fee validation can lead to users losing their unclaimed fees when executing this instruction.

Recommendation

Implement a validation to ensure that the unclaimed fee of the LP is zero before allowing deletion.

Mitigation Review Log

Fixed in commit bcabf13376307fe9aed3630e0d12a1d2a4b6d60b.

4.4 Lack of Slippage Protection for Price and PT Amount in Earn Invest

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `earn_invest` instruction lacks slippage protection to safeguard users against sudden price or liquidity changes during execution.

Additionally, when the swap reaches its price bound, only a portion of the input margin is utilized, which may lead to unintended behavior.

Impact

Without slippage protection, users may mint PT at an unfavorable rate, resulting in suboptimal outcomes.

If the swap hits the price bound, only a portion of the input amount is used. This behavior may be unexpected if the user wants an FOK-style request.

Recommendation

Introduce a `min_pt_amount` parameter to provide slippage protection.

Mitigation Review Log

Fixed in commit cdbb633f46f1b640f4ba3c474ba9c3fda1fd6ae3.

4.5 Zero Division May Cause DoS in Liquidation

Severity: Low	Status: Fixed
Target: Smart Contract	Category: DoS Risk

Description

In the `handle_liquidate` function, the calculation of the trade price can lead to a zero division error if `base_amount_transfered` is zero. The problematic line is:

```
200 let trade_price =
    ↪ mul_div_i64(quote_amount_transfered.safe_add(margin_to_quote)?.safe_sub
201 (liq_fee_quote)?, ONE_I64, -base_amount_transfered)?;
```

programs/ratex-contracts/src/instructions/liquidation.rs#L200-L200

When `base_amount_transfered` equals zero, this will cause the liquidation process to fail.

A correct implementation, as seen in the `handle_liquidate_insurance` function, is as follows:

```
229     let trade_price = if base_amount_transfered != 0 {
230       mul_div_i64(quote_amount_transfered, ONE_I64, -base_amount_transfered)?
231   } else {
232       0
233   };
```

programs/ratex-contracts/src/instructions/liquidation_insurance.rs#L229-L233

Impact

If `base_amount_transfered` is zero, the liquidation will fail, potentially leading to a denial of service.

Recommendation

Implement the logic from `handle_liquidate_insurance` to prevent zero division errors and ensure the robustness of the liquidation process.

Mitigation Review Log

Fixed in commit b09c8a1974d6ec69fdb9f4878dc5b37aada9d094.

4.6 Residual Input Amount Return Can Panic end_vault_swap IX

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `end_vault_swap` instruction, if `is_exact_in` is true, the instruction checks `amount_in` must be exactly equal to the parameter `amount`, which is the `in_margin_market.flash_loan_amount` when entering the `end_vault_swap` instruction.

```
253         validate!(
254             amount_in == amount,
255             ErrorCode::InvalidJupSwap,
256             "amount_in != amount",
257         )?;
```

programs/ratex-contracts/src/instructions/swap.rs#L253-L257

The issue is that, if there is any residual input token left in the `in_user_token_account`, it will be returned to the vault and deducted from `amount_in`. The `amount_in` will never be `amount` (i.e. `in_margin_market.flash_loan_amount`) in this case. The instruction will panic in this check.

programs/ratex-contracts/src/instructions/swap.rs#L206-L220

Impact

For certain routes, exact in mode cannot guarantee full consumption of all input tokens. This results in repeated failures of VaultSwap related instructions under such conditions.

Recommendation

The check should be `amount == in_margin_market.flash_loan_amount` .

Mitigation Review Log

Fixed in commit e37f15946c0b0716322eb6a1b5082e7f5c116cbc.

4.7 Price on Tick Boundary May Cause Zero Liquidity for Unified Position Calculation

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the implementation of the unified position feature, the code utilizes `yield_market.pool.liquidity` to determine the liquidity of the unified position. For example:

```
173     let settled_base_amount = if yield_market.pool.liquidity == 0 {
174         0
175     } else {
176         mul_div_i128(
177             liquidity_amount.cast()?,
178             unified_position.settled_base_amount.cast()?,
179             yield_market.pool.liquidity.cast()?
180         )?
181     };
```

programs/ratex-contracts/src/instructions/lp.rs#L173-L181

However, when the price of the AMM pool hits the boundary of the upper price tick, `yield_market.pool.liquidity` can become zero, leading to a potential zero division error during calculations.

Impact

This corner case, where the price reaches the tick boundary, can cause `yield_market.pool.liquidity` to be zero. This would adversely affect the settled amount calculation and the epoch update. Although the likelihood of this situation occurring is low due to the constraints of `min_order_size` and `order_step_size` in the yield market, it remains a concern.

Recommendation

Implement an additional state variable to monitor the liquidity of the unified position independently from `yield_market.pool.liquidity` or introduce safeguards to prevent the price from reaching the tick boundary

Mitigation Review Log

Fixed in commit a38352fb6af591d00afe210006b6b86195fc7d50

and f02c264a547d8d2fe759d161f9d577cd4b6539ab.

4.8 Precision Loss in Yield Position Fee Calculation

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Precision Issue

Description

In the implementation of the yield position fee feature, the fee for each position is calculated during the position rebase using floor division:

```
252 let quote_rebased_on_protocol = mul_div_i64(quote_rebased,
    ↪  YIELD_POSITION_FEE_RATIO.cast()?, PERCENT_PRECISION.cast()?)?;
```

programs/ratex-contracts/src/state/lp.rs#L252-L252

However, the protocol fee is updated in `unified_rebase` by calculating the portion of total `quote_rebased` . This approach may introduce a precision issue when the recorded protocol fee exceeds the sum of the protocol fees charged for each position.

Impact

The precision loss can lead to dust bad debt. While there are other calculations that may offset this precision loss, it is advisable to address the root cause to prevent potential issues.

Recommendation

Use ceil division in the position rebase calculation.

Mitigation Review Log

Fixed in commit 91e53be14cd7fd423402c5a150a1b019dbf62f95.

4.9 Informational and Undetermined Issues

Lack of Validation for UserAuthority in get_user_and_token_account Function

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Data Validation

The `get_user_and_token_account` function loads three accounts: the User account, the token account, and the user authority account. These accounts will be used in the `margin_withdraw_on_close` function. However, it doesn't check whether the user authority is equal to the authority of the User account.

Lack of Validation for other_amount_threshold in end_vault_swap IX

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Data Validation

The `begin_vault_swap` and `end_vault_swap` instructions are used to swap different margins for rebalancing. However, they do not validate whether the `other_amount_threshold` is a reasonable amount according to the oracle. This is concerning because a keeper has significantly higher privileges in these instructions than other keeper instructions, which could lead to potential vault losses.

5 Disclaimer

This report reflects the security status of the project as of the date of the audit. It is intended solely for informational purposes and should not be used as investment advice. Despite carrying out a comprehensive review and analysis of the relevant smart contracts, it is important to note that Offside Labs' services do not encompass an exhaustive security assessment. The primary objective of the audit is to identify potential security vulnerabilities to the best of the team's ability; however, this audit does not guarantee that the project is entirely immune to future risks.

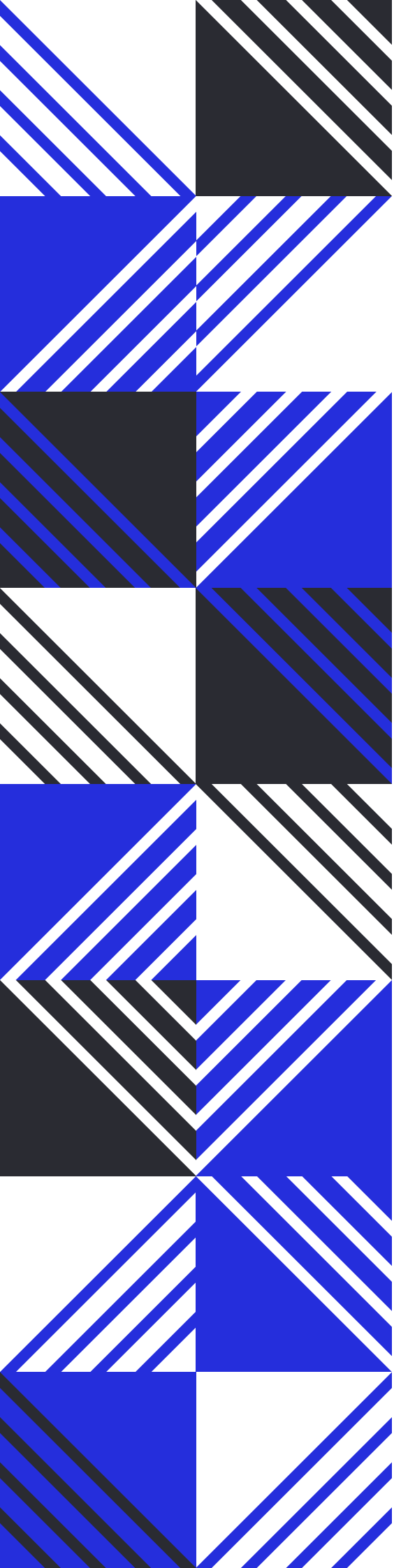
Offside Labs disclaims any liability for losses or damages resulting from the use of this report or from any future security breaches. The team strongly recommends that clients undertake multiple independent audits and implement a public bug bounty program to enhance the security of their smart contracts.

The audit is limited to the specific areas defined in Offside Labs' engagement and does not cover all potential risks or vulnerabilities. Security is an ongoing process, regular audits and monitoring are advised.

Please note: Offside Labs is not responsible for security issues stemming from developer errors or misconfigurations during contract deployment and does not assume liability for centralized governance risks within the project. The team is not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By utilizing this report, the client acknowledges the inherent limitations of the audit process and agrees that the firm shall not be held liable for any incidents that may occur after the completion of this audit.

This report should be considered null and void in case of any alteration.



OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs