OFFSIDE LABS

RateX

Smart Contract Security Assessment

September 2024

Prepared for:

Rate Labs Limited

Prepared by:

Offside Labs

Yao Li

Ripples Wen

Ronny Xing

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.


https://offside.io/


https://github.com/offsidelabs


https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Ratex* smart contracts, starting on July 22, 2024, and concluding on September 10, 2024.

Project Overview

RateX is a decentralized exchange on Solana that specializes in leveraged yield trading. It allows users to trade synthetic Yield Tokens with leverage. Additionally, users can earn fixed yield through synthetic Principal Tokens.

Audit Scope

The assessment scope contains mainly the smart contracts of the *ratex-contracts* program for the *Ratex* project.

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

- Ratex
  - Branch: dev-audit
  - Commit Hash: 88e57a61f2681264725585a61269d7bd0f80d6a8
  - Codebase Link

We listed the files we have audited below:

- Ratex
  - programs/ratex-contracts/src/*.rs

Findings

The security audit revealed:

- 0 critical issue
- 2 high issues
- 6 medium issues
- 5 low issues
- 6 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	Lack of Constraint for Tick Array Accounts in Remaining Accounts	High	Fixed
02	Multiple Active Yield Positions With market_index = 0 Are Possible	High	Fixed
03	Lack of Constraint for Depsoit Margin Market	Medium	Fixed
04	Close Yield Position for Isolated User May Fail	Medium	Fixed
05	Incorrect Math Formula in liquidate_insurance Instruction	Medium	Fixed
06	Incorrect Market Status Check Prevents LP and Earn from Exiting After Maturity	Medium	Fixed
07	Insurance Yield Position Can Not Be Liquidated When base_asset_amount Is Zero	Medium	Fixed
08	Abnormal Prices May Break Invariants During Epoch Updates	Medium	Fixed
09	Close Account May Fail	Low	Fixed
10	Liquidation Does Not Check the Market Status	Low	Fixed
11	Precision Remainder Exists in social_loss_yield_position Before Reset	Low	Fixed
12	PnL calculation Is Incorrect When Opening Opposite Position	Low	Fixed
13	Insurance Yield Position Will Be Lost After Yield Market Expiration	Low	Fixed
14	Lack of Validation for admin Account in MultiSigDepsoit Instruction	Informational	Acknowledged
15	Lack of Constraint for mint_account Accounts in EpochUpdate Instruction	Informational	Fixed
16	Lack of Constraint for token_vault Accounts in RemoveLiquidity Instruction	Informational	Fixed
17	Lack of Constraint for user_token_account Account in CancelIsolatedOrder Instruction	Informational	Fixed
18	Lack of Constraint for user and user_stats Accounts in Liquidate Instruction	Informational	Fixed
19	Swap Fee Can Be Skipped in One Tick	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Lack of Constraint for Tick Array Accounts in Remaining Accounts

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

Description

In some instructions, `tick_array_accounts` need to be loaded from remaining accounts and utilized for swap operations. However, there is no validation to ensure that the `ammpool` field of these accounts matches the expected yield market's `ammpool`.

Impact

The same issue exists in multiple instructions. Their code locations are as follows:

- Earn:
  - programs/ratex-contracts/src/instructions/earn.rs#L110-L114
  - programs/ratex-contracts/src/instructions/earn.rs#L269-L274
- LP: programs/ratex-contracts/src/instructions/lp.rs#L446-L451
- User: programs/ratex-contracts/src/instructions/user.rs#L513-L517

A malicious attacker could potentially substitute `tick_array_accounts` from a different AMM pool. When searching for the next tick with liquidity during the swap process, it will return an index controlled by the attacker. This could result in swaps being executed with incorrect pricing and fee distribution, leading to unintended profits for the attacker.

Recommendation

Implement validation to ensure that the `ammpool` field of `tick_array_accounts` matches the yield market's `ammpool` before proceeding with the swap operation.

Mitigation Review Log

RateX Team:

This issue was identified and has since been resolved. The constraints for tick array accounts in remaining accounts have been properly implemented to ensure stability and security.

- Relevant code implementation

Offside Labs: Fixed. The mitigation has added check for ammpool pubkey in `load` function.

4.2 Multiple Active Yield Positions With market_index = 0 Are Possible

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

Description

The default value for the `market_index` of `YieldPosition` is 0. And the implementation of `user.get_yield_position_index(market_index)` returns the first yield position with a matching `market_index` . Consequently, if `state.market_index_start` is set to 0, a user can exploit this by:

1. Open a position with `market_index = 1` .
2. Open another position with `market_index = 0` .
3. Close the position with `market_index = 1` .
4. Open another position with `market_index = 0` .

As a result, the user can hold two active yield positions with `market_index = 0` .

Impact

This issue can lead to several problems:

- **Yield Market Expiry Settlement:** It can interfere with the correct settlement of yield market expiry.

  Reference: src/instructions/admin.rs#L601-L601

- **Yield Market Open Interest Update:** If the first position is an empty position, it will affect the update of open interest in the yield market.

  Reference: src/controller/orders.rs#L231-L231

- **Incomplete Liquidation:** This can lead to incomplete liquidation processes.

  Reference: src/instructions/liquidation_insurance.rs#L84-L84

Recommendation

Fix the implementation of `get_yield_position_index` to properly handle this case, or ensure that `state.market_index_start` is always greater than 0 to prevent this issue from occurring.

Mitigation Review Log

RateX Team:

The problem allowing multiple active yield positions with `market_index = 0` has been addressed and resolved in the latest code submission. This fix ensures that only one position can be active under these conditions.

- Relevant code implementation

Offside Labs: Fixed.

4.3 Lack of Constraint for Depsoit Margin Market

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

Description

In the `deposit` and `multi_sig_deposit` instructions, `margin_index` can be set by users.

```
718     #[account(
719         mut,
720         constraint = margin_market.load()?.market_index == market_index,
721     )]
722     pub margin_market: AccountLoader<'info, MarginMarket>,
```

programs/ratex-contracts/src/instructions/user.rs#L718-L722

However, the `withdraw` instruction is currently restricted to only withdraw from the first margin market.

Impact

If a user deposits fund into a margin market without `margin_index_start` , and then attempts to withdraws, the user may experience a loss of funds.

Additionally, other users may encounter withdrawal failures due to insufficient funds in the first margin market.

Recommendation

Impose a constraint such that `margin_index = state.margin_index_start` for both the `deposit` and `multi_sig_deposit` instructions.

Mitigation Review Log

RateX Team:

This issue has been resolved in the codebase, ensuring that deposit margin markets now have the appropriate constraints for secure transactions.

- Relevant code implementation

Offside Labs: Fixed.

4.4 Close Yield Position for Isolated User May Fail

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

Description

If a user is set as `user.isolated = true`, and holds more than one yield position, any subsequent attempt to close a yield position may fail.

Impact

There is currently no validation restricting an isolated user from holding multiple yield positions. However, when an isolated user attempts to close a single yield position, the entire margin is withdrawn due to the following codes. This can cause a collateral check failure, leading to the failure of the entire instruction.

Relevant code references:

- src/controller/orders.rs#L292
- src/instructions/user.rs#L560

As a result, if an isolated user attempts to fully close one of multiple yield positions, the collateral check may fail, causing the entire close position order to fail.

Recommendation

Fix the validation logic for isolated users.

Mitigation Review Log

RateX Team:

A check has been introduced to prevent isolated users from opening more than one yield position. If an isolated user tries to open multiple positions, an error will be triggered.

- Relevant code implementation

Offside Labs: Fixed.

4.5 Incorrect Math Formula in liquidate_insurance Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Math

Description

In the `liquidate_insurance` instruction, the `base_asset_amount_before` is incorrectly calculated as `base_amount_held - base_amount_filled` by the following code

```
151 let base_asset_amount_before =
    ↪ base_amount_held.safe_sub(maker_liquidation_stat.base_amount_filled)?;
```

programs/ratex-contracts/src/instructions/liquidation_insurance.rs#L151

However, the current base amount held by the position is updated by the following code:

```
143     maker_position.update_position(
144         base_amount_filled.saturating_neg(),
145         quote_amount_filled
146             .saturating_neg()
147             .safe_sub(maker_realized_pnl)?,
148     )?;
```

programs/ratex-contracts/src/controller/liquidation.rs#L143-L148

It is equivalent to `base_amount_held = base_asset_amount_before - base_amount_filled`. So the correct calculation should use `safe_add` instead of `safe_sub`.

Impact

This incorrect calculation could lead to inaccurate updates of active user positions in the yield market.

When the admin updates the epoch, the number of active positions will be checked as an important invariant. An incorrect count of active positions may cause some positions to be skipped during the epoch update, or it may cause the epoch update to fail, resulting in a continuous suspension of the protocol.

Recommendation

Correct the formula by replacing `safe_sub` with `safe_add` to ensure accurate calculations in the `liquidate_insurance` instruction.

Mitigation Review Log

RateX Team:

The incorrect math formula in the liquidate_insurance instruction has been fixed, resolving this issue in the codebase.

Offside Labs: Fixed.

4.6 Incorrect Market Status Check Prevents LP and Earn from Exiting After Maturity

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Timing

Description

The exiting instructions of LP and Earn have the following checks for the markets status:

1. `earn_redeem` programs/ratex-contracts/src/instructions/earn.rs#L254
2. `remove_lp_shares` programs/ratex-contracts/src/instructions/lp.rs#L356-L359

```
validate! {
    yield_market.is_active(now)?,
    ErrorCode::YieldMarketPaused
}?;

// yield_market.is_active
pub fn is_active(&self, now: i64) -> RatexResult<bool> {
    ...
    let not_expired = self.expire_ts == 0 || now < self.expire_ts;
    Ok(status_ok && not_expired)
}
```

The check for `not_expired` ensures the market is not mature.

Impact

The `yield_market.is_active` check prevents LP and Earn from Exiting After Maturity.

Recommendation

Exit instructions should still be allowed to be called normally after the market matures.

Mitigation Review Log

RateX Team:

This issue has been resolved, allowing LPs and Earn participants to exit after maturity without market status issues.

Offside Labs: Fixed.

4.7 Insurance Yield Position Can Not Be Liquidated When base_asset_amount Is Zero

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

Description

If the current `insurance_yield_position` has `base_asset_amount = 0`, the `liquidate_insurance` instruction will be panic and exit when calculating the `break_even_price` in the `check_insurance_coverage` function:

```
32 let break_even_price = mul_div_i64(quote_amount, ONE_I64,
   ↪ base_amount.saturating_neg())?;
```

programs/ratex-contracts/src/controller/liquidation.rs#L32

Moreover, the check before entering social loss transfer also prevents the liquidation of insurance positions.

```
181 if yield_market.insurance_yield_position.base_asset_amount != 0 &&
    ↪ adl_finish {
```

programs/ratex-contracts/src/instructions/liquidation_insurance.rs#L181

Impact

If the current `insurance_yield_position` has `base_asset_amount = 0` and `quote_asset_amount < 0`, it is a debt needs to be liquidated and transfered to the `social_loss_yield_position`.

Since the insurance cannot be liquidated, this portion of bad debt will continue to remain in the protocol without being distributed until `insurance_yield_position.base_asset_amount` is no longer 0.

Recommendation

Once the ADL for traders is finished, if the `insurance_yield_position` is with `base_asset_amount = 0` and `quote_asset_amount < 0`, it should be transfered into social loss directly.

Mitigation Review Log

RateX Team:

This issue has been fixed.

- Relevant code implementation

Offside Labs: Fixed.

4.8 Abnormal Prices May Break Invariants During Epoch Updates

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

Description

When the admin performs an epoch update on the yield markets, the yield markets will first be paused, at which point all trading will be suspended. However, if the AMM prices severe fluctuations due to certain factors(for example, front run, huge mis-trading without slippage protection, black swan, etc.) before the market is paused, causing the `yield_market.pool.sqrt_price` at the time of the epoch update to deviate significantly from the TWAP price, then some critical invariants may be broken after the update.

Impact

We only list two impacts, but there may be other invariants affected by this issue as well.

1. TWAP price manipulation:

   Under normal cases, manipulating the TWAP of a Uni V3 type CLMM is considered impossible, as the attacker would incur enormous costs. However, since the epoch update needs to iterate through all LPs and Users to adjust positions, this process could potentially last for several minutes. Especially in cases where the Solana network is unstable, this process might take even longer.

   If the AMM prices are abnormal when the markets are paused. This price will be retained for several minutes and will be written into the `ObservationState` at the next swap.

   Once this duration exceeds the `twap_duration` (default is 5 minutes), the TWAP price could potentially be manipulated.

2. Incorrect OV Calculation:

   Because OV calculation `calculate_ov` is based on the current sqrt price of the Amm pool, programs/ratex-contracts/src/controller/epoch_update.rs#L225-L246 , an abnormal price may lead to incorrect LP position adjustments for OV.

Recommendation

1. Time accumulation in `ObservationState` should be also paused when the market is paused.
2. It would be a good idea to add an admin/protocol position that can adjust the price in the opposite direction through swap when the market is paused and the current price unreasonably deviates from the TWAP. In this case, the attacker's attack cost will become profit for the protocol.

Mitigation Review Log

RateX Team:

To prevent breaking invariants due to abnormal prices, we will adjust the system to prohibit liquidations for 5 minutes following an epoch update.

- Relevant code implementation

Offside Labs: Fixed.

4.9 Close Account May Fail

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

Description

In the `delete_user` instruction, the account `user` will be closed and the remaining SOL will be transferred to account `authority`. However, the `authority` is not marked with `mut` in the definition.

```
668 pub struct DeleteUser<'info> {
669     #[account(
670         mut,
671         has_one = authority,
672         close = authority
673     )]
674     pub user: AccountLoader<'info, User>,
675     ...
676     pub authority: AccountInfo<'info>,
677 }
```

programs/ratex-contracts/src/instructions/user.rs#L668-L688

The `delete_lp` instruction has a similar issue.

Impact

The instruction may fail if `payer` and `authority` are not the same account. Since there is a constraint `payer.key() == authority.key() || keepers.keepers.contains(&payer.key())` for this instruction, the `payer` may be one of keepers by design.

It's important to note that since the fee payer is treated as mutable automatically, the failure will not occur when the `authority` is set as the fee payer.

Proof of Concept

Call `delete_user` instruction with `payer == keepers[0]` and `payer != authority`, using `payer` as transaction fee payer. The call will fail with an error indicating `instruction changed the balance of a read-only account`.

Recommendation

Add `mut` notation to the account `authority`.

Mitigation Review Log

RateX Team:

This issue has been resolved, ensuring that accounts can now be closed without encountering errors.

- Relevant code implementation

Offside Labs: Fixed.

4.10 Liquidation Does Not Check the Market Status

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Timing

Description

Theoretically, liquidation can occur at any stage, even after the market matures. However, liquidation for traders should not happen during epoch updates. This could potentially lead to liquidating users who are in an intermediate state during epoch updates, resulting in serious accounting errors.

The instruction `liquidate` does not check if the market or the liquidating user is in the process of epoch update.

Recommendation

Add checks for the market status or users status.

Mitigation Review Log

RateX Team:

A check for `yield_market.is_active(now)` has been added to ensure liquidation actions are only performed when the market is active.

- Relevant code implementation

Offside Labs: Fixed.

4.11 Precision Remainder Exists in social_loss_yield_position Before Reset

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

Description

`social_loss_yield_position` will be reset directly in the `epoch_update_end` instruction. However, according to the distribution approch of the bad debt in the `epoch_update_remove` instruction, there is always precision remainder existing in the `social_loss_yield_position` .

```
882 let lp_ratio = mul_div_i128(lp_base_quota.cast::<i128>()?, ONE_I128,
    ↪ total_lp_base_quota.cast::<i128>()?)?;
883
884 let sloss_quote_amount = mul_div_i128(sloss_quote_amount.cast::<i128>()?,
    ↪ lp_ratio, ONE_I128)?.cast::<i64>()?;
885 let sloss_base_amount = mul_div_i128(sloss_base_amount.cast::<i128>()?,
    ↪ lp_ratio, ONE_I128)?.cast::<i64>()?;
```

programs/ratex-contracts/src/instructions/admin.rs#L882-L885

Social loss amount filled by LPs will be truncated as `i64` . So there are always some bad debt amounts left in the position.

Impact

The precision remainder of bad debt brought by each epoch update is related to the number of LPs, and will be approximately half the number of LPs. These precision remainders will continuously accumulate with each epoch update. However, they are directly reset to zero in the accounting.

This portion of bad debt is not evenly distributed among all LPs, which will result in the last LPs to exit being unable to withdraw normally and forced to bear all the remaining bad debt.

Recommendation

The distribution of bad debt can be changed to round up, which can ensure that the overall collateralization ratio of the protocol is sufficient. However, this may result in some precision remainders in the vault at the end.

Mitigation Review Log

RateX Team:

We use the `mul_div_i128_ceil` method to accurately calculate the amount allocated to each LP, ensuring precision in the distribution process.

- Relevant code implementation

Offside Labs: Fixed.

4.12 PnL calculation Is Incorrect When Opening Opposite Position

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

Description

Trader and liquadation IXs use `calculate_realized_pnl` function to calculate the PnL when reducing the position. For opposite opening, it uses the following pseudocode to get the part of the `quote_amount_filled` belonged to the original position.

```
filled_ratio = | base_asset_amount / base_amount_filled |
close_quote_amount = filled_ratio * quote_amount_filled
PnL = close_quote_amount + quote_asset_amount
```

programs/ratex-contracts/src/state/user.rs#L532

The issue is that, due to limited liquidity, the change in swap prices has slippages. The original average price of the closing position portion is lower/higher, but it needs to be averaged with the opening price.

Proof of Concept

Suppose the initial position is that, -50 quote and 100 base. And based on the current liquidity, user can sell 100 base for 50 quote, or sell 200 base for 60 quote.

If the user first close the inital position and then open a new opposite position, the PnL will be 0, and the opposite position will be 10 quote and -100 base.

But if the user fills a order with -200 base to open a opposite position directly, he will get a -30 quote PnL and an opposite position with 30 quote and -100 base.

Impact

This PnL calculation forces the user's margin balance to be reduced and converted into a quote position, which causes the overall collateralization ratio to decrease.

Recommendation

It is more recommended to divide the opposite opening process into two segments for calculating PnL.

Mitigation Review Log

RateX Team:

This has been resolved by disabling opposite positions.

- Relevant code implementation

Offside Labs: Fixed.

4.13 Insurance Yield Position Will Be Lost After Yield Market Expiration

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

Description

If an insurance yield position holds a positive quote amount and has not been liquidated by the time the yield market expires, there is currently no mechanism to handle the remaining funds.

Impact

The funds held in the insurance yield position will be permanently lost, as there is no entry point to recover or manage these assets after the yield market has expired.

Recommendation

Implement logic to handle insurance yield positions when the yield market expires, ensuring that any remaining funds are appropriately managed or returned.

Mitigation Review Log

RateX Team:

A function has been added to allow admin to claim insurance positions after the yield market expires, preventing any loss of these positions.

- Relevant code implementation

Offside Labs: Fixed.

4.14 Informational and Undetermined Issues

Lack of Validation for admin Account in MultiSigDepsoit Instruction

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

In the `multi_sig_deposit` instruction, there is an additional signer, referred to as the admin. However, this admin account currently serves no meaningful purpose and is not validated.

Mitigation Review Log:

RateX Team: Acknowledged. The mainnet version will replace this method with a new AML verification method.

Lack of Constraint for mint_account Accounts in EpochUpdate Instruction

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

In the `EpochUpdate` instruction, there is currently no validation to ensure that the `mint_account_a` and `mint_account_b` accounts provided in the instruction struct match the `quote_asset_vault` and `base_asset_vault` of the yield market.

If `asset_vault` and `token_vault` are mistakenly used interchangeably, it may cause the protocol to cease functioning.

Mitigation Review Log:

Offside Labs: Fixed.

Lack of Constraint for token_vault Accounts in RemoveLiquidity Instruction

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

In the `RemoveLiquidity` instruction, there is currently no validation to ensure that the `token_vault_base` and `token_vault_quote` accounts provided in the instruction struct match the `token_vault_base` and `token_vault_quote` of the yield market's AMM pool.

Mitigation Review Log:

Offside Labs: Fixed.

Lack of Constraint for user_token_account Account in CancelIsolatedOrder Instruction

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

In the `CancelIsolatedOrder` instruction, there is currently no validation to ensure that the authority of `user_token_account` matches the `user.authority` .

Mitigation Review Log:

Offside Labs: Fixed.

Lack of Constraint for user and user_stats Accounts in Liquidate Instruction

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

It is necessary to validate the same authority for `user` and `user_stats` accounts in instruction struct `Liquidate` . Currently, `user_stats` is unused in the instruction.

Mitigation Review Log:

Offside Labs: Fixed.

Swap Fee Can Be Skipped in One Tick

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Logic Error

Amm swap has the following separate branches for calculating fees when `a_to_b = false`

```
198         if amount_specified_is_input && !is_max_swap {
199             fee_amount = amount_remaining - amount_in;
200         }
```

programs/ratex-contracts/src/math/amm/swap_math.rs#L198-L200

The key condition triggering this branch is that `is_max_swap` is false, meaning the input amount specified for the current swap cannot reach the limit price or the edge of the current tick. At this case, the following codes adjust `amount_in` to match the actual next price:

```
149         get_next_sqrt_price(
150             sqrt_price_current,
151             liquidity,
152             amount_calc,
153             amount_specified_is_input,
154             a_to_b,
155         )?
```

programs/ratex-contracts/src/math/amm/swap_math.rs#L149-L155

```
169     if !is_max_swap {
170         amount_fixed_delta = get_amount_fixed_delta(
171             sqrt_price_current,
172             next_sqrt_price,
173             liquidity,
174             amount_specified_is_input,
175             a_to_b,
176         )?;
177     }
```

programs/ratex-contracts/src/math/amm/swap_math.rs#L169-L177

However, since we didn't adjust the `amount_calc` for fees when calling `get_next_sqrt_price` , then `amount_calc = amount_remaining = amount_fixed_delta = amount_in` . The fee amount will be zero.

Users can control the input amount to skip fees within one tick, potentially causing the protocol and LPs to lose a significant amount of fees.

Since all swap fees in the protocol are collected externally to the swap and do not participate in the internal input amount adjustment of the swap, this branch can be deleted.

Mitigation Review Log:

RateX Team:

The current implementation of the swap supports only two modes:

- a_to_b = true, amount_specified_is_input = true
- a_to_b = false, amount_specified_is_input = false .

As a result, the yt amount is always used as the exact input/output, meaning the branch of code that could skip the swap fee will never be triggered.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

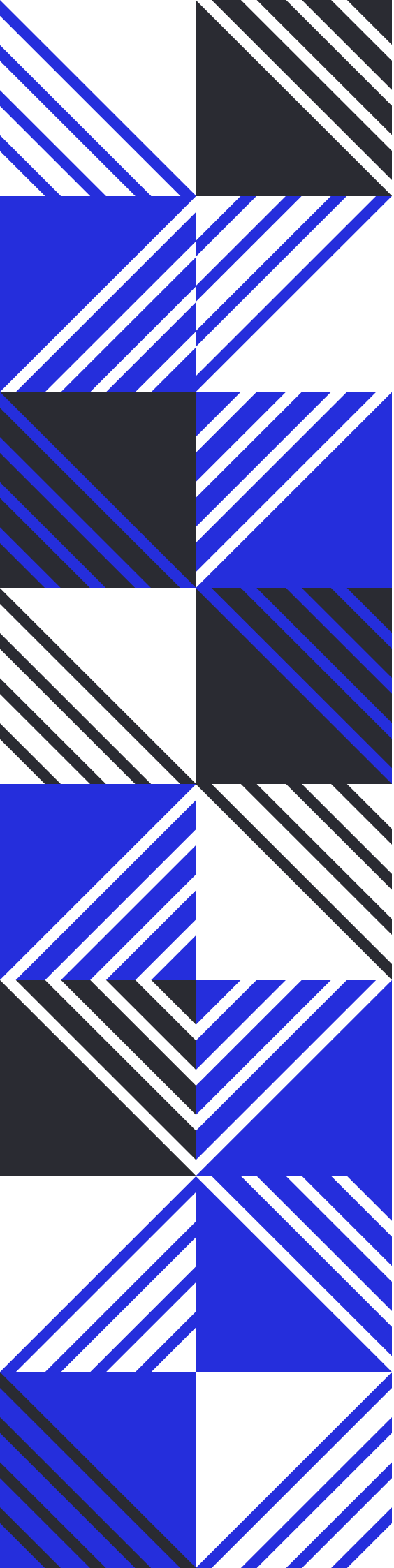
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.



OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs