OFFSIDE LABS

# Exponent Core

## Smart Contract Security Assessment

**September 2024**

**Prepared for:**

**Exponent Finance**

**Prepared by:**

**Offside Labs**

*Ripples Wen*

*Siji Feng*

# 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 *Exponent* smart contracts, starting on *October 7th, 2024*, and concluding on *October 16th, 2024*.

### Project Overview

The Core program facilitates yield-based swaps for yield-bearing DeFi positions on Solana, allowing users to trade fixed and floating yield exposures. It divides these positions into:

1. Principal Token (PT): Redeemable for the underlying asset at maturity.
2. Yield Token (YT): Allows holders to claim yield until maturity. Key features include:
    - Stripping: Converts positions into PT and YT, holding the original in escrow.
    - Staking: YT holders earn yield and additional incentives.
    - Merging: Recombines PT and YT into the original position; post-maturity, PT redeems 1:1 for the base token.

The vault module manages the lifecycle of these tokens, enhancing yield management without frequent adjustments. An AMM supports trading PT with its correlated SY token, and LP tokens can earn farming rewards and SY yield. Additionally, an Admin program features a "super admin" who sets keys for different "principles", storing the list of admins for specific actions, ensuring efficient management and control within the program.

### Audit Scope

The assessment scope contains mainly the smart contracts of the *Exponent Core & Admin* program for the *Exponent Core* project.

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

- *Exponent Core*
    - Codebase: https://github.com/exponent-finance/exponent-core
    - Commit Hash: 78bb3be42dcbf31662d13ce7a735a41bf4c32850

We listed the files we have audited below:

- *Exponent Core*
    - solana/programs/exponent_core
    - solana/programs/exponent_admin
    - solana/libraries/precise_number
    - solana/libraries/sy_common
    - solana/libraries/tcurve
    - solana/libraries/anchor_util
    - solana/libraries/token_util
    - solana/libraries/amount_value

### Findings

The security audit revealed:

- 1 critical issues
- 1 high issues
- 1 medium issues
- 0 low issue
- 10 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 PT/SY Balance Management in sell_-yt/buy_yt Instructions | Critical | Fixed |
| 02 | Potential Overflow Issue in trade_pt::handler Sanity Check | High | Fixed |
| 03 | Lack of Ownership Constraints in stage_yt_yield and deposit_yt Instructions | Medium | Fixed |
| 04 | remove_principle_admin Resizing Issue on Missing admin_to_remove | Informational | Acknowledged |
| 05 | Potential Inequity in Treasury Fees for Vault Interests and Emissions | Informational | Acknowledged |
| 06 | Inaccurate Comment in deposit_yt::validate Method | Informational | Fixed |
| 07 | Redundant Check for Vault in handle_withdraw_yt() | Informational | Fixed |
| 08 | Inaccurate Comment on Vault's sy_for_pt Definition | Informational | Fixed |
| 09 | Inconsistent Surplus Distribution After Maturity for YT Holders | Informational | Acknowledged |
| 10 | Optimization for current_rate_scalar and current_rate_anchor | Informational | Partially Fixed |
| 11 | Variable Shadowing Issue in add_liquidity() | Informational | Partially Fixed |
| 12 | Inefficient Runtime Calculation of discriminator in CPI | Informational | Fixed |
| 13 | Replacing Division with Multiplication | Informational | Partially Fixed |

# 4 Key Findings and Recommendations

## 4.1 Incorrect PT/SY Balance Management in sell_yt/buy_yt Instructions

| Severity: Critical | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: Inconsistent State |

### Description

In the `sell_yt` instruction, PT is flash-borrowed from the market and merged with YT to acquire SY. The SY is then sold in the open market through a CPI call to `trade_pt` in order to obtain PT for repaying the flash loan.

```
205     // First, borrow an amount of PT equal to YT
206     ctx.accounts.market.financials.dec_pt_balance(yt_in);
207 ...
208     // perform the purchase of PT with the SY
209     let sy_spent = ctx
210         .accounts
211         .do_cpi_buy_pt(
212             ctx.remaining_accounts,
213             yt_in.try_into().expect("overflow converting yt_in to i64"),
214             sy_constraint,
215         )
216         .expect("Trade PT failed");
217 
218     // must reload the market to get the updated financials
219     ctx.accounts.market.reload()?;
220 ...
221     // Re-adjust the market's balance of PT
222     ctx.accounts.market.financials.inc_pt_balance(yt_in);
```

solana/programs/exponent_core/src/instructions/market_two/sell_yt.rs#L205-L257

However, during the PT borrowing process, the market's PT balance in the `MarketFinancials` struct is reduced, but this change is not serialized back to the account data. As a result, the CPI call to `trade_pt` utilizes the original PT balance, rather than the updated, decreased balance. This leads to an incorrect PT/SY exchange rate in `trade_pt`.

Moreover, after the `trade_pt` call, the `MarketFinancials` struct is reloaded from the account data, effectively discarding the previous balance reduction. Consequently, when the flash loan is repaid, the PT balance in the `MarketFinancials` struct is incorrectly increased to an erroneous value.

A similar bug exists in the `buy_yt` instruction, where the SY balance reduction not being properly serialized leads to incorrect calculations in subsequent operations.

Furthermore, in wrapper instructions, a similar issue with self-CPI exists. For example, in `wrapper_buy_yt`, after calling `self_cpi::do_cpi_buy_yt`, the vault account is modified but not reloaded. As a result, when `wrapper_buy_yt` finishes, the unmodified data is saved to the vault account on-chain, overwriting the changes made by the self-CPI.

#### Recommendation

Serialize the reduced PT/SY balance back to the account data before making the CPI call to `trade_pt`, ensuring that the exchange rate is calculated correctly and that subsequent operations reflect the accurate balance.

## 4.2 Potential Overflow Issue in trade_pt::handler Sanity Check

| Severity: High | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: DOS |

#### Description

In the sanity check of `trade_pt::handler`, we assert that `net_trader_sy` and `net_trader_pt` have opposite signs, though the comment contradicts the implementation. We check for opposite signs by multiplying the two i64 variables. However, if the multiplication overflows—such as when both values are around the maximum u32—the program will panic in a release build due to the overflow.

```
185     // sanity check
186     // net_trader_sy and net_trader_pt must have the same sign
187     assert!(
188         trade_result.net_trader_sy * net_trader_pt <= 0,
189         "Invalid trade result -- SY change and PT change must have
            ↪ opposite signs"
190     );
```

solana/programs/exponent_core/src/instructions/market_two/trade_pt.rs#L185-L190

#### Impact

The potential for overflow can lead to program panics, particularly when trading amounts are significant. Note that the mint decimals of PT and SY usually correspond to the underlying base token. Stablecoins like USDC and PYUSD have a decimal of 6, which means we could trigger the overflow by trading around 5000. For tokens with greater decimals, the panic seems inevitable.

### Recommendation

Use a simple `if/else` pattern to check for opposite signs instead of relying on multiplication. Additionally, update the comment to align with the implementation for clarity.

### Mitigation Review Log

Exponent Team: Fixed in relevant code implementation

Offside Labs: `Fixed`

## 4.3 Lack of Ownership Constraints in stage_yt_yield and deposit_yt Instructions

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

### Description

In the `stage_yt_yield` instruction, both the `user_yield_position` and the vault robot's `yield_position` are updated. However, the `user_yield_position` is not constrained to be owned by the user, which allows the vault robot's `yield_position` to be mistakenly passed as the `user_yield_position`. In this scenario, the vault robot's `yield_position` could be earned twice. Since the data between these two updates is not synchronized, this duplication of earnings will lead to incorrect values in the vault.

A similar issue exists in the `deposit_yt` instruction, where the lack of constraints on the ownership of the `user_yield_position` can also result in unintended duplications and inaccuracies in the vault's recorded values.

### Recommendation

To address this, while it is expected to allow `stage_yt_yield` and `deposit_yt` operations for non-owners, it is necessary to add a check to ensure that the passed `user_yield_position` is not the same as the vault robot's `yield_position`. This will prevent the potential for duplicated earnings and maintain the integrity of the vault's values.

### Mitigation Review Log

Exponent Team: Fixed in relevant code implementation

Offside Labs: `Fixed`

## 4.4 Informational and Undetermined Issues

### remove_principle_admin Resizing Issue on Missing admin_to_remove

| Severity: Informational | Status: Acknowledged |
| --- | --- |
| Target: Smart Contract | Category: Unhandled Failure |

In the instruction `remove_principle_admin`, it does not revert immediately when the `admin_to_remove` is not found. However, at the end of this instruction, it attempts to resize the Admin account unconditionally by 32 bytes. Fortunately, the result won't be saved because the serialization fails. Note: Normally, incorrect invocation of privileged instructions should not be a concern; this issue was just a leftover from a previous audit. It is unclear whether the serialization failure is a desired outcome.

### Potential Inequity in Treasury Fees for Vault Interests and Emissions

| Severity: Informational | Status: Acknowledged |
| --- | --- |
| Target: Smart Contract | Category: Edge Case |

Treasury fees for vault interests and emissions are charged to users upon collection. However, the admin has the ability to modify vault settings using `AdminAction::ChangeVaultBpsFee` and `AdminAction::ChangeEmissionBpsFee`, allowing for changes to the fee rate at any time. This raises the possibility of inequity among users, as they could experience differing fee rates.

### Inaccurate Comment in deposit_yt::validate Method

| Severity: Informational | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: Documentation Issue |

The comment in `deposit_yt::validate` states, "only allow amount 0 deposits" after maturity, but it actually rejects all deposits.

### Redundant Check for Vault in handle_withdraw_yt()

| Severity: Informational | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: Redundant Code |

The check `vault.is_active(now)` in `handle_withdraw_yt()` is redundant, as expired vaults are already rejected during the initial validation.

### Inaccurate Comment on Vault's sy_for_pt Definition

| Severity: Informational | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: Documentation Issue |

The comment on the definition of `Vault`'s `sy_for_pt` states that "this value stops changing after the vault is expired". However, it is updated and should be recalculated every time the `sy_exchange_rate` is updated.

### Inconsistent Surplus Distribution After Maturity for YT Holders

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

After maturity, for YT holders, the number of staged uncollected SY remains constant over time. This implies that any surplus value generated by these SY belongs to the YT holders. However, the surplus emissions associated with these SY are currently being dispatched to the lambo fund instead of being distributed to the YT holders. This creates an inconsistency.

```
337 // check if vault will collect treasury emissions from post-maturity
    ↪ emissions
338
339 if self.can_collect_emission_lambo(now) {
340     let surplus_emissions = self.calc_emission_surpluses(&sy_state);
341     self.increase_emission_lambo_fund(surplus_emissions);
342 }
```

solana/programs/exponent_core/src/state/vault.rs#L337-L341

### Optimization for current_rate_scalar and current_rate_anchor

| Severity: Informational | Status: Partially Fixed |
| --- | --- |
| Target: Smart Contract | Category: Optimization |

The `current_rate_scalar` and `current_rate_anchor` are always used together, and we call `current_rate_scalar()` when calculating `current_rate_anchor()`. We could simply supply the `current_rate_scalar` value as a parameter to the `current_rate_anchor()` function. These functions are not utilized outside of the `trade_pt()` function, so they could be inlined without losing simplicity. Additionally, both functions use `sec_remaining(now)`, which could also be simplified by storing it as a local variable.

```
374     /// Calculate the current rate anchor
375     fn current_rate_anchor(&self, sy_exchange_rate: Number, now: u64) ->
        ↪ DNum {
376         let sec_remaining = self.sec_remaining(now);
377         let asset = self.asset_balance(sy_exchange_rate).floor_u64();
378         let current_rate_scalar = self.current_rate_scalar(now);
```

solana/programs/exponent_core/src/state/market_two.rs#L374-L378

### Variable Shadowing Issue in add_liquidity()

| Severity: Informational | Status: Partially Fixed |
| --- | --- |
| Target: Smart Contract | Category: Optimization |

In `add_liquidity()`, we convert numbers from u64 to f64 for higher precision. However, the original variables are shadowed because we reused their names, which prevents access to the original values (e.g., `intent_sy` and `intent_pt`). To avoid an unnecessary conversion from f64 back to u64, we should use different variable names.

```
142     let intent_sy = N::from_u64(intent_sy);
143     let intent_pt = N::from_u64(intent_pt);
```

solana/libraries/tcurve/src/math.rs#L142-L143

### Inefficient Runtime Calculation of discriminator in CPI

| Severity: Informational | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: Optimization |

We use `anchor_util::ix_discriminator` to calculate the discriminator for self-cpis just in time. However, this runtime calculation involves unnecessary string operations and a syscall to sha256. These values are actually constant and could be calculated statically.

```
1  pub fn ix_discriminator(name: &str) -> [u8; 8] {
2      let preimage = format!("global:{}", name);
3      let mut sighash = [0u8; 8];
4      sighash.copy_from_slice(
5          &anchor_lang::solana_program::hash::hash(preimage.as_bytes())
6          .to_bytes()[..8],
7      );
8      sighash
9  }
```

solana/libraries/anchor_util/src/lib.rs#L1-L8

### Replacing Division with Multiplication

| Severity: Informational | Status: Partially Fixed |
| --- | --- |
| Target: Smart Contract | Category: Optimization |

In DeFi protocols on EVM, there is no preference for multiplication over division, as both MUL and DIV have the same gas usage for native uint256 numbers. However, high-precision computation on Solana requires embedding external mathematical libraries, where the cost of division operations is significantly higher than that of multiplication. As a result, replacing divisions with multiplications can save computing units. For example, `rate_scalar` is only used in the denominators of the formulas for `rate_anchor` and `exchange_rate`, and it's calculated from `rate_scalar_root` /

`normalized_sec_remaining`. We can simply use the inverse of `rate_scalar` instead. This approach also allows us to handle division by zero exceptions naturally.

```rust
/// The rate scalar grows as the seconds remaining decrease
///
/// The sensitivity of the curve has an inverse relationship to the rate scalar
/// As the rate scalar grows, the sensitivity goes down
pub fn rate_scalar<N: Num>(rate_scalar_root: N, sec_remaining: u64) -> N {
    // if sec_remaining is zero, the rate scalar is effectively infinite
    if sec_remaining == 0 {
        return N::max();
    }

    let normalized_sec_remaining = normalized_sec_remaining::<N>(sec_remaining);

    // root * YEAR_SEC / sec_remaining
    rate_scalar_root / normalized_sec_remaining
}
```

solana/libraries/tcurve/src/math.rs#L88-L102

# 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.

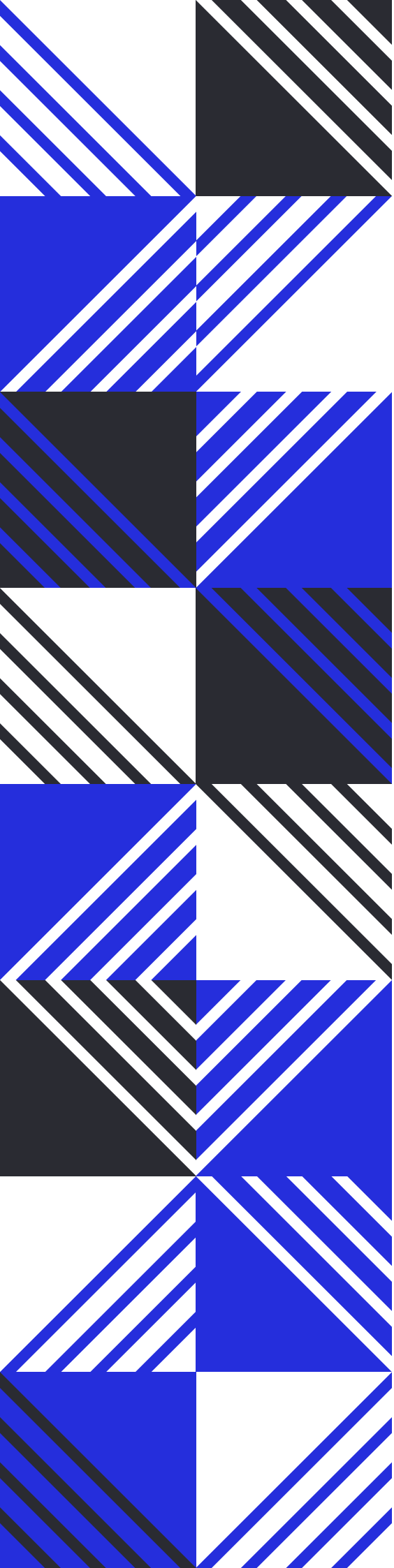
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster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