OFFSIDE LABS

# Exponent Generic Standard

## Smart Contract Security Assessment

**February 2025**

**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 February 11th, 2025, and concluding on February 14th, 2025.

### Project Overview

The Generic Standard program enables users to mint standard yield tokens with other yield tokens. Key functionalities include:

1. mint_sy and redeem_sy: Manage deposits into escrow and handle minting or redeeming of standard yield tokens. The deposited tokens are other yield tokens, and their value is measured in base tokens like USDC and SOL.
2. Emissions Management: Accrues emissions for staked receipt tokens, facilitating easier tracking for the core program. Non-staked emissions are directed to the protocol's treasury.

### Audit Scope

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

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

- *Exponent*
  - Codebase: `https://github.com/exponent-finance/exponent-core`
  - Commit Hash: 769235113cea11fb6a1f0613cf62b79d5bbca3a4

We listed the files we have audited below:

- *Exponent*
  - solana/programs/generic_standard/src/*.rs
  - solana/libraries/fragmetric_cpi/src/*.rs
  - solana/libraries/pyth_v2_cpi/src/*.rs

### Findings

The security audit revealed:

- 0 critical issue
- 1 high issues
- 0 medium issue
- 1 low issues
- 0 informational issue

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 Meta Account Restriction in get_position | High | Fixed |
| 02 | Prevent Multiple Price Updates Within Same Slot for Pyth Oracle | Low | Fixed |

# 4 Key Findings and Recommendations

## 4.1 Lack of Meta Account Restriction in get_position

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

### Description

```rust
#[account(
    mut,
    realloc = Position::size_of(meta.emissions.len()),
    realloc::payer = meta,
    realloc::zero = true,
)]
pub position: Account<'info, Position>,

#[account(
    mut,
    has_one = mint_sy,
    has_one = token_sy_escrow,
)]
pub meta: Box<Account<'info, SyMeta>>,
```

solana/programs/generic_standard/src/instructions/read/get_position.rs#L11-L24

In the `get_position` instruction, the meta account is not restricted to match the one recorded in the position account. This allows an attacker to call the instruction with a mismatched position and meta account, potentially corrupting the data stored in the position account.

### Impact

```rust
Some(reward) => {
    assert_eq!(
        reward.mint, emission.mint,
        "Reward mint does not match emission tracker index"
    );
}
```

solana/programs/generic_standard/src/state/position.rs#L26-L31

If the `get_position` instruction is executed with a position account that contains emission reward information and a meta account with fewer emissions, the recorded mint in the position's emission reward data may be corrupted. As a result, any attempt to withdraw rewards from this position would fail, leading to potential fund loss for the position

owner.

```
73 let share_index_delta = share_index
74     .checked_sub(&self.last_seen_share_index)
75     .unwrap();
76
77 let reward_balance_to_stage = share_index_delta
78     .checked_mul(&Number::from_natural_u64(sy_balance))
79     .unwrap()
80     .floor_u64();
81
82 self.claimable_rewards_amount = self
83     .claimable_rewards_amount
84     .checked_add(reward_balance_to_stage)
85     .unwrap();
86
87 self.last_seen_share_index = share_index;
```

solana/programs/generic_standard/src/state/position.rs#L73-L87

If two meta accounts exist with the same emissions, calling `get_position` with an mismatched position and meta account may artificially inflate the `last_seen_share_index` and `claimable_rewards_amount`. An attacker could exploit this inconsistency to claim emissions that do not belong to them, resulting in financial losses for the contract.

#### Recommendation

Enforce a strict relationship between the position account and the associated meta account by adding the `has_one = meta` Anchor constraint to the position account.

#### Mitigation Review Log

Fixed in the commit 09b217106390da43651627bae55524eb64f5d723.

## 4.2 Prevent Multiple Price Updates Within Same Slot for Pyth Oracle

| Severity: Low | Status: Fixed |
| --- | --- |
| Target: Smart Contract | Category: Arbitrage |

#### Description

In the existing implementations of standard programs, the exchange rate between SY and base tokens are presumed to be stable inside one transaction: neither `mint_sy` or `redeem_sy` could change the exchange rate.

However, this is not true for the `Pyth` based exchange rate oracle. A `Pyth` price feed can be updated whenever it's fed with a valid price update by anyone. The `Pyth` price can fluctuate inside one transaction, so does the `get_index()` for `Pyth`.

#### Proof of Concept

If the price can be manipulated in this transaction, a user can leverage risk-less flash loan to arbitrage the price difference. For example, if the exchange rate oracle goes down significantly (greater than the `conf_interval`), a user can:

1. Borrow base tokens
2. Mint SY tokens
3. Strip SY to PT + YT
4. Move the price down
5. Merge PT + YT to SY
6. Redeem SY to base tokens
7. Repay the base tokens

Since PT worths more SY in step 5 than in step 3, the user has surplus SY tokens and thus surplus base tokens (Mint/Redeem SY is not affected by the exchange rate oracle).

If the exchange rate oracle goes up, the user can still profit if they have SY as debt.

#### Recommendation

Record a `current_slot` of the last pyth price we update the `current_index`. If it's the current slot, we just return the `current_index` (This value is different from the `posted_slot` from `Pyth`'s `PriceUpdateV2`).

#### Mitigation Review Log

Fixed in the commit d16eefeb0cb03dd79b49b01d267b30fa2dde3466 and 84dda49a21f0a4940ec059e28cf0efba7ea9d4ea.

# 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.

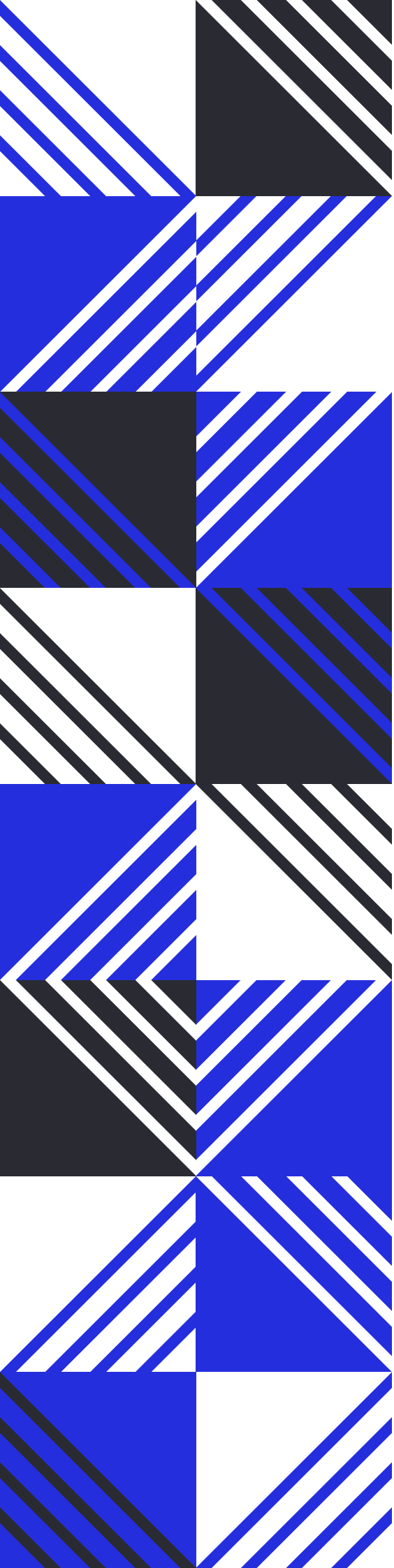
We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster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