OtterSec

Exponent Generic Standard

Security Assessment

February 22nd, 2025 — Prepared by OtterSec

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Exponent Finance engaged OtterSec to assess the `generic-standard` program. This assessment was conducted between February 17th and February 19th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 9 findings throughout this audit engagement.

In particular, we identified a high-risk vulnerability where it is possible for emissions to be removed, disrupting the integrity of reward tracking in positions, resulting in a failure in consistency checks and halting protocol operations (OS-EGS-ADV-00), and another issue concerning incorrect length calculation and utilization of an inefficient method to convert to byte slice (OS-EGS-ADV-03). We also highlighted a discrrepency in the check for ignoring negative fluctuations within the confidence interval, which utilizes a greater than comparison instead of a less than comparison (OS-EGS-ADV-01).

Furthermore, the hook modification instruction utilizes fixed lengths from the meta account to reallocate memory for hook discriminators, instead of dynamically utilizing the input hook's length, potentially resulting in incorrect memory allocation (OS-EGS-ADV-04).

We also made recommendations to ensure adherence to coding best practices (OS-EGS-SUG-01) and advised incorporating additional checks within the codebase for improved robustness and security (OS-EGS-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/exponent-finance/exponent-core. This audit was performed against commit 28d615b.

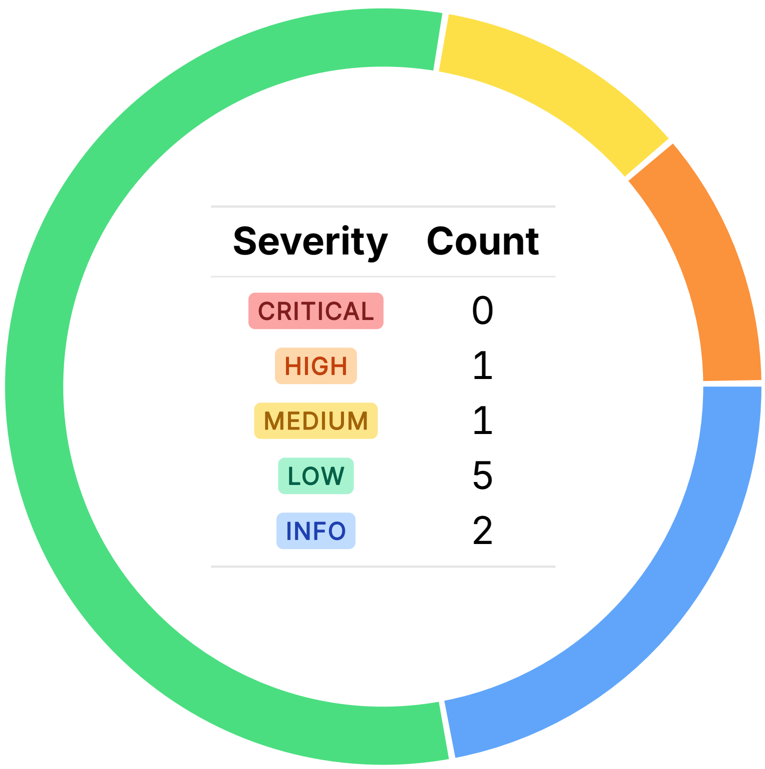
A brief description of the program is as follows:

Name	Description
generic-standard	It serves as a lightweight interface for the Exponent Core program, primarily handling wrapped yield-bearing tokens (SY tokens).

03 — Findings

Overall, we reported 9 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	1
MEDIUM	1
LOW	5
INFO	2

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-EGS-ADV-00	HIGH	RESOLVED ⊘	It is possible for emissions to be removed, disrupting the integrity of reward tracking in positions, resulting in failures in consistency checks and halting protocol operations.
OS-EGS-ADV-01	MEDIUM	RESOLVED ⊘	In the Pyth interface, the check for ignoring negative fluctuations within the confidence interval is incorrect due to the utilization of a greater-than comparison.
OS-EGS-ADV-02	LOW	RESOLVED ⊘	The `MintSy` and `RedeemSy` instructions inconsistently utilize token program interfaces.
OS-EGS-ADV-03	LOW	RESOLVED ⊘	In the `SyMeta` instruction, the `SyMeta::LEN_STATIC` calculation is incorrect.
OS-EGS-ADV-04	LOW	RESOLVED ⊘	The `ModifyHook` instruction utilizes fixed lengths from the meta account to reallocate memory for hook discriminators instead of dynamically utilizing the input hook's length, potentially resulting in incorrect memory allocation.
OS-EGS-ADV-05	LOW	RESOLVED ⊘	The `token_vrt_escrow` account currently utilizes `token_program`, restricting the yield-bearing token to the standard token program.

OS-EGS-ADV-06	LOW	RESOLVED	Currently, the code relies on the mutable `hook.enabled` flag to update `interface_emissions_accounts_until`, which may result in state inconsistencies if `hook.enabled` is modified mid-execution.

Violation of Protocol Integrity via Emission Removal HIGH OS-EGS-ADV-00

Description

The vulnerability concerns the potential for an emission to be removed from `SyMeta` in a way that breaks the integrity of the system. The `Position` structure tracks the state of a user's position, including their amount and a list of rewards, which are tied to emissions. Each `Reward` in `Position` corresponds to an `Emission` on `SyMeta`, tracked by its `mint` and `last_seen_share_index` which is saved in `Reward`. `Position::ensure_trackers` ensures that the position has a corresponding reward entry for every emission that exists.

>_ src/state/position.rs RUST

```rust
pub fn ensure_trackers(&mut self, emissions: &Vec<Emission>) {
    for (index, emission) in emissions.iter().enumerate() {
        match self.reward_indexes.get_mut(index) {
            Some(reward) => {
                assert_eq!(
                    reward.mint, emission.mint,
                    "Reward mint does not match emission tracker index"
                );
            }[...]
        }
    }
}
```

The `RemoveEmission` operation enables removing an emission from `SyMeta`, shifting the remaining emissions. In the context of `Position::ensure_trackers`, if an emission is removed from the list, then all subsequent emissions will shift by one index, and as the mint is already saved on all the user positions' rewards, the reward mint will not match the corresponding emission tracker index and the function will fail. The exponent-core relies on the order of rewards in the position. If emissions are removed and the order is disrupted, the protocol will no longer safely calculate rewards or track their distribution, disrupting the entire protocol, resulting in a denial-of-service for all positions.

Remediation

Disallow the `RemoveEmission` operation.

Patch

Resolved in #1914.

Improper Handling of Negative Fluctuation MEDIUM OS-EGS-ADV-01

Description

The issue lies in the way the code handles negative fluctuations in the exchange rate when comparing the `new_exchange_rate` to the `current_index`. Specifically, When utilizing the Pyth price feed to retrieve the latest price in `Utils::get_index`, the check for ignoring negative fluctuations within the confidence interval is incorrect. Negative fluctuations that are within the confidence interval should be ignored, but the current logic accepts negative fluctuations even if they are above the confidence interval.

>_ src/utils.rs RUST

```rust
pub fn get_index<'a>(meta: &SyMeta, remaining_accounts: &'a [AccountInfo]) -> Result<Number> {
    match meta.interface_type {
        InterfaceType::Pyth => {
            [...]
            // The price account has very small negative fluctuations of less than 2 price units
                ↪  because of publishers that post slightly outdated indices.
            // Ignore negative fluctuations as long as they are within confidence interval,
                ↪  which is very tight for redemption index feeds.
            if new_exchange_rate < current_index
                && current_index.checked_sub(&new_exchange_rate).unwrap() > conf_interval
            {
                return Ok(current_index);
            }
            Ok(new_exchange_rate)
        }[...]
    }
}
```

Remediation

Update the comparison to ensure the negative fluctuations are below the confidence interval.

Patch

Resolved in #1915.

Inconsistent Token Program Utilization LOW OS-EGS-ADV-02

Description

The vulnerability in the `MintSy` and `RedeemSy` instructions arises from improper utilization of the `token_base_program` and token_program. In both the `RedeemSy` and `MintSy` (shown below) instructions, the `token_vrt_escrow` account is utilized in conjunction with `token_base_program` only during transfers. However, the `token_vrt_escrow` account is defined utilizing `token_program`. `token_base_program` should be utilized for token program checks of `token_vrt_escrow`.

>_ src/instructions/mint_sy.rs RUST

```rust
#[derive(Accounts)]
pub struct MintSy<'info> {
    [...]
    /// SY robot's base token account
    #[account(
        mut,
        associated_token::authority = meta,
        associated_token::mint = meta.yield_bearing_mint,
        associated_token::token_program = token_program,
    )]
    [...]
    pub token_vrt_escrow: InterfaceAccount<'info, TokenAccount>,
    pub base_token_program: Interface<'info, TokenInterface>,
    pub token_program: Interface<'info, TokenInterface>,
}

impl<'info> MintSy<'info> {
    /// Transfer base token to account owned by SY robot
    fn ctx_transfer_base(&self) -> CpiContext<'_, '_, '_, 'info, Transfer<'info>> {
        CpiContext::new(
            self.base_token_program.to_account_info(),
            Transfer {
                from: self.token_base_depositor.to_account_info(),
                to: self.token_vrt_escrow.to_account_info(),
                authority: self.depositor.to_account_info(),
            },
        )
    }
    [...]
}
```

Also, in the `IinitSy` instruction, `mint_sy` is directly initialized utilizing a `Token` program. Thus, the `token_program` should not be treated as `TokenInterface`.

Remediation

Ensure `MintSy` and `RedeemSy` instructions utilize `token_base_program` for the `token_vrt_escrow` account checks.

Patch

Resolved in #1916.

Discrepancy in Size Calculation LOW OS-EGS-ADV-03

Description

The `SyMeta::LEN_STATIC` calculation is incorrect due to a mismatch between the expected and actual memory size. This occurs as the current implementation of `SyMeta::LEN_STATIC` calculation fails to include all the variables on `SyMeta` as part of the length calculation, resulting in incomplete length calculation.

Remediation

Update the `LEN_STATIC` length calculation to ensure that it properly accounts for all variables in `SyMeta`.

Patch

Resolved in #1915.

Incorrect Hook Length Allocation LOW OS-EGS-ADV-04

Description

Currently, in `ModifyHook`, the reallocation of the `SyMeta` account is based on the lengths of `pre_mint_hook_discriminator` and `post_redeem_hook_discriminator` within the `meta` account. The reallocating logic looks at the current lengths of these fields in the meta account to determine how much space needs to be allocated. If the new hook configuration has a different length for these discriminators, the meta account's reallocation may not be appropriate.

>_ src/instructions/admin/modify_hook.rs RUST

```rust
#[derive(Accounts)]
pub struct ModifyHook<'info> {
    [...]
    #[account(
        mut,
        // realloc to make room for one new emission
        realloc = SyMeta::len(meta.emissions.len(), meta.interface_accounts.len(),
            ↪ meta.hook.pre_mint_hook_discriminator.len(),
            ↪ meta.hook.post_redeem_hook_discriminator.len()),
        realloc::payer = fee_payer,
        realloc::zero = false,
    )]
    pub meta: Account<'info, SyMeta>,
    [...]
}
```

Remediation

Utilize the lengths of the hook discriminators from the input `hook` passed as part of the instruction, rather than from the `meta` account.

Patch

Resolved in #1913.

Unnecessary Restriction of Yield-Bearing Token LOW OS-EGS-ADV-05

Description

The `InitSy` instruction initializes a synthetic yield-bearing token (`SY` token) by setting up various accounts, including a `token_vrt_escrow` account. The issue arises because the `token_vrt_escrow` account is explicitly tied to `token_program`, which is assumed to be the standard SPL Token program (standard token program). However, some yield-bearing tokens may not be issued under this program. By enforcing the utilization of `token_program`, the instruction unintentionally restricts the yield-bearing token to only those utilizing SPL Token, preventing compatibility with yield-bearing tokens from token-2022 program.

>_ src/instructions/admin/init_sy.rs RUST

```rust
pub struct InitSy<'info> {
    [...]
    #[account(
        associated_token::authority = sy_meta,
        associated_token::mint = yield_bearing_mint,
        associated_token::token_program = token_program,
    )]
    pub token_vrt_escrow: InterfaceAccount<'info, TokenAccount>,
    [...]
}
```

Remediation

Add an additional `base_token_program` account. This allows the escrow account to use a flexible token program, ensuring compatibility with different yield-bearing token standards.

Patch

Resolved in #1912.

State Modification via Mutable Flag Value LOW OS-EGS-ADV-06

Description

In `AddEmission` and `RemoveEmission`, there is potential state inconsistency issue due to the dynamic and mutable nature of `hook.enabled`. Both `AddEmission` and `RemoveEmission` instructions update `hook.interface_emissions_accounts_until` depending on whether `hook.enabled` is true at the time of execution. However, `hook.enabled` is mutable and may be toggled via the `ModifyHook` instruction mid-execution, which may incorrectly update `hook.interface_emissions_accounts_until`, resulting in improper emission account handling.

>_ *src/instructions/admin/remove_emission.rs* RUST

```rust
impl RemoveEmission<'_> {
    [...]
    pub fn remove_emission(&mut self, index: u8) {
        self.meta.emissions.remove(index as usize);

        if self.meta.hook.enabled {
            self.meta.hook.interface_emissions_accounts_until = self
                .meta
                .hook
                .interface_emissions_accounts_until
                .checked_sub(1)
                .unwrap();
        }
    }
}
```

Remediation

Prevent the updating of `hook.enabled` by `ModifyHook`, or ensure `interface_emissions_accounts_until` is not updated based on the value of `hook.enabled`.

Patch

Acknowledged by the Exponent Finance development team.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-EGS-SUG-00	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-EGS-SUG-01	Suggestions regarding ensuring adherence to coding best practices.

Missing Validation Logic

OS-EGS-SUG-00

Description

1. The `ModifyHook` instruction does not validate whether the `meta.interface_type` is set to `Fragmetric`, even though the hook being modified is intended only for the `Fragmetric` interface type.
2. In `Utils::get_index`, the `price.exponent` is utilized to scale the price and confidence interval. In the `InterfaceType::Pyth` case, it is assumed to always be negative. However, if `price.exponent` were unexpectedly positive, it would result in incorrect scaling.

Remediation

1. Add a validation step to ensure that the `meta.interface_type` is `Fragmetric`.
2. Ensure that `price.exponent` is negative before processing to enforce expected behavior and mitigate unexpected scenarios.

Code Maturity OS-EGS-SUG-01

Description

1. In the `InitSy::validate`, the current approach ties admin validation to `principles.jito_restaking`. It would be more appropriate to utilize separate principles for generic standard admin instructions.

   >_ src/instructions/admin/init_sy.rs RUST

   ```rust
   pub fn validate(&self) -> Result<()> {
       self.admin_state
           .principles
           .jito_restaking
           .is_admin(&self.admin.key)?;
       Ok(())
   }
   ```

2. Utilize `interface_type.to_bytes` instead of the `INTERFACE_BYTES` array for converting the `interface_type` into a byte slice within `SyMeta::authority_seeds`.

Remediation

Implement the above-mentioned suggestions.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
HIGH	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
MEDIUM	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
LOW	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
INFO	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.