Exponent Jito Restaking

Security Assessment

November 31st, 2024 — Prepared by OtterSec

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

Executive Summary **2**

Overview 2

Key Findings 2

Scope **3**

Findings **4**

Vulnerabilities **5**

OS-EXF-ADV-00 | Invariant Fragility 6

OS-EXF-ADV-01 | Inconsistency in Admin Update Mechanism 7

General Findings **8**

OS-EXF-SUG-00 | Code Maturity 9

OS-EXF-SUG-01 | Missing Validation Logic 10

Appendices

Vulnerability Rating Scale **11**

Procedure **12**

01 — Executive Summary

Overview

Exponent Finance engaged OtterSec to assess the `jito-restaking-integration` program. This assessment was conducted between December 18th and November 20th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 4 findings throughout this audit engagement.

In particular, we have identified a critical vulnerability in the fragility of the invariant check linking the token-vrt-escrow balance to the mint-sy supply, allowing a single SY token burn to result in DoS of the protocol (OS-EXF-ADV-00).

Lastly, in the SY initialization instruction, the exponent admin is designated as the update authority for the metadata. As a result, if the admin changes in the future, all previously created SY metadata instances will continue to have the previous admin as their update authority (OS-EXF-ADV-01).

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

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/exponent-finance/exponent-core/tree/fix-kysol-market-calc. This audit was performed against commit 6a6581c.

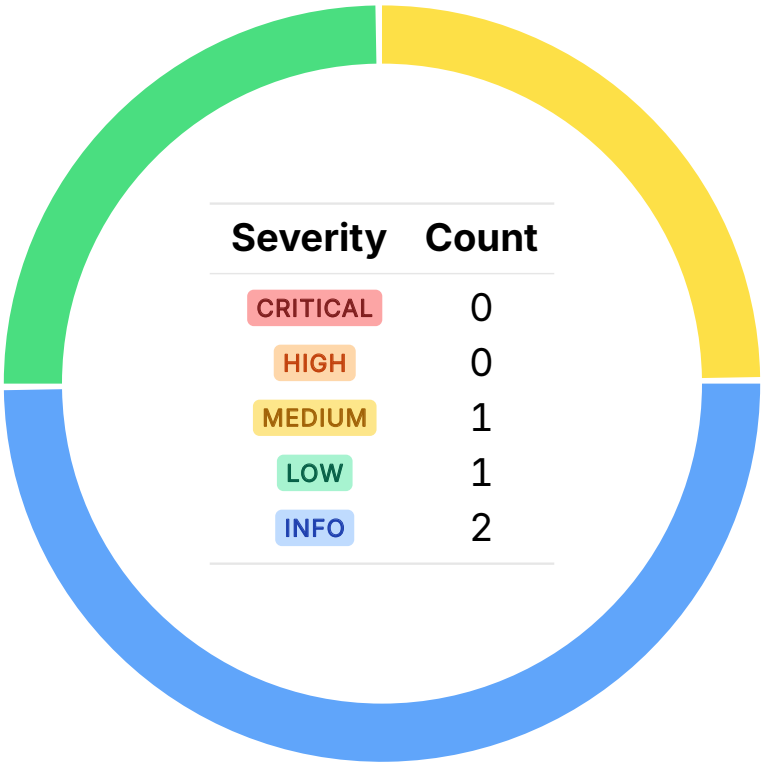
A brief description of the program is as follows:

Name	Description
jito-restaking-integration	The audit focused on the integration of SY programs, the interface for retrieving the SPL stake pool exchange rate, and the jito-restaking-cpi, which includes the state for the Jito restaking program and its associated helper functions.

03 — Findings

Overall, we reported 4 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	0
MEDIUM	1
LOW	1
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-EXF-ADV-00	MEDIUM	RESOLVED	The invariant check linking `token_vrt_escrow` balance to `mint_sy` supply is fragile, allowing a single `SY` token burn to result in failure, exposing the protocol to potential denial-of-service attacks.
OS-EXF-ADV-01	LOW	RESOLVED	The `InitSy` instruction sets a static update authority for metadata tied to the initial admin, resulting in a vulnerability where updates to `SY` metadata remain under the control of the previous admin even after administrative changes.

Invariant Fragility MEDIUM OS-EXF-ADV-00

Description

In the `MintSy` and `RedeemSy` instructions, the `invariant` is utilized to validate that the amount of base tokens in the `token_vrt_escrow` account equals the supply of synthetic yield (`SY`) tokens in the `mint_sy` account to ensure that `SY` is backed by enough `VRT`. However, in the case that even a single `SY` token is burned, the `mint_sy.supply` decreases, and consequently, the check in the `invariant` would fail. The invariant failure prevents any subsequent mints or redemptions, effectively creating a denial of service scenario.

>_ jito_restaking_standard/src/instructions/redeem_sy.rs RUST

```rust
fn invariant(&mut self) -> Result<()> {
    invariant(&self.token_vrt_escrow.amount, &self.mint_sy.supply)
}
```

Remediation

Replace strict equality with an acceptable range or a margin of error.

Patch

Fixed in bbca170.

Inconsistency in Admin Update Mechanism LOW OS-EXF-ADV-01

Description

In `InitSy`, the admin calling the instruction is set as the `update_authority` for the newly created metadata. This link is permanent for that specific `SY` instance. In the future, if the exponent admin changes, all previously created `SyMeta` instances will retain the old admin as their `update_authority`. Thus, older tokens remain tied to the old admin, resulting in disjoint governance and operational complexity. This is particularly dangerous if the original admin is compromised, as it renders the protocol vulnerable even if it is updated with a new admin.

Remediation

Ensure the migration of the `update_authority` for all existing metadata accounts in the event of admin transitions.

Patch

Acknowledged by the developers.

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-EXF-SUG-00	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.
OS-EXF-SUG-01	There are several instances where proper validation is not done, resulting in potential security issues.

Code Maturity OS-EXF-SUG-00

Description

1. Utilize `init_if_needed` for the `token_vrt_escrow` account in the `InitSy` instruction to ensure that the account is created only if it does not already exist.

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

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

2. Currently, Jito Restaking utilizes `marginfi_standard` from `admin_state` for all admin instructions. Thus, those with administrative control over MarginFi automatically gain access to Jito Restaking's admin functions unnecessarily, rendering it difficult to establish granular controls specific to Jito Restaking. Create a new set of principles specific to Jito Restaking to ensure that admin control over Jito Restaking remains independent.

Remediation

Implement the above-mentioned suggestions.

Missing Validation Logic OS-EXF-SUG-01

Description

1. The absence of mint and authority validation for token accounts introduces risks, as these checks ensure token accounts are correctly configured with the appropriate token mint and authority. Explicitly validate the token mint and authority.
2. `get_index` does not validate whether the `stake_pool`'s state is up-to-date by checking the `last_update_epoch`. Exchange rate calculation relies on up-to date `stake_pool` data. If the `stake_pool` is stale, the exchange rate returned by the function may be incorrect, resulting in inaccurate index values. Include a staleness check on the `StakePool` utilizing the `last_update_epoch`.

>_ jito_restaking_interface_spl_stake_pool/src/lib.rs RUST

```rust
pub fn get_index(ctx: Context<GetIndex>) -> Result<JitoRestakingInterfaceReturnData> {
    let stake_pool: StakePool =
        borsh1::try_from_slice_unchecked(&ctx.accounts.stake_pool.data.borrow()).unwrap(;
    let virtual_exchange_rate = Number::from_natural_u64(stake_pool.total_lamports)
        .checked_div(&Number::from_natural_u64(stake_pool.pool_token_supply))
        .unwrap();
    Ok(JitoRestakingInterfaceReturnData {
        index: virtual_exchange_rate,
    })
}
```

Remediation

Incorporate the missing validations into the codebase.

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.