



Solana Liquid ETF Staking

Security Assessment

July 29th, 2024 — Prepared by OtterSec

Ajay Shankar Kunapareddy

d1r3wolf@osec.io

Robert Chen

notdeghost@osec.io

Table of Contents

Executive Summary	2
Overview	2
Key Findings	2
Scope	2
Findings	3
Vulnerabilities	4
OS-SRP-ADV-00 Incorrect Account Check	5
General Findings	6
OS-SRP-SUG-00 Centralization Risk	7
OS-SRP-SUG-01 Code Validation	8
Appendices	
Vulnerability Rating Scale	9
Procedure	10

01 — Executive Summary

Overview

Solana Liquid ETF Staking engaged OtterSec to assess the **restaking** program. This assessment was conducted between April 22nd and April 25th, 2024. We did a followup review on July 24th, 2024. For more information on our auditing methodology, refer to [Appendix B](#).

Key Find

We produced 3 findings throughout this audit engagement.

In particular, we identified a high-risk vulnerability concerning the freeze account functionality, which checks the frozen status of the wrong account ([OS-SRP-ADV-00](#)). Furthermore, we identified possible additional checks during token initialization if the supply of the **RST** token mint is a non-zero amount ([OS-SRP-SUG-00](#)), and also highlighted the need for additional verification in the token account ([OS-SRP-SUG-01](#)).

Scope

The source code was delivered to us in a Git repository at <https://github.com/solana-liquid-etf-staking/restaking-program>. This audit was performed against commit [ad83447](#). We did a followup review for PR [#7](#).

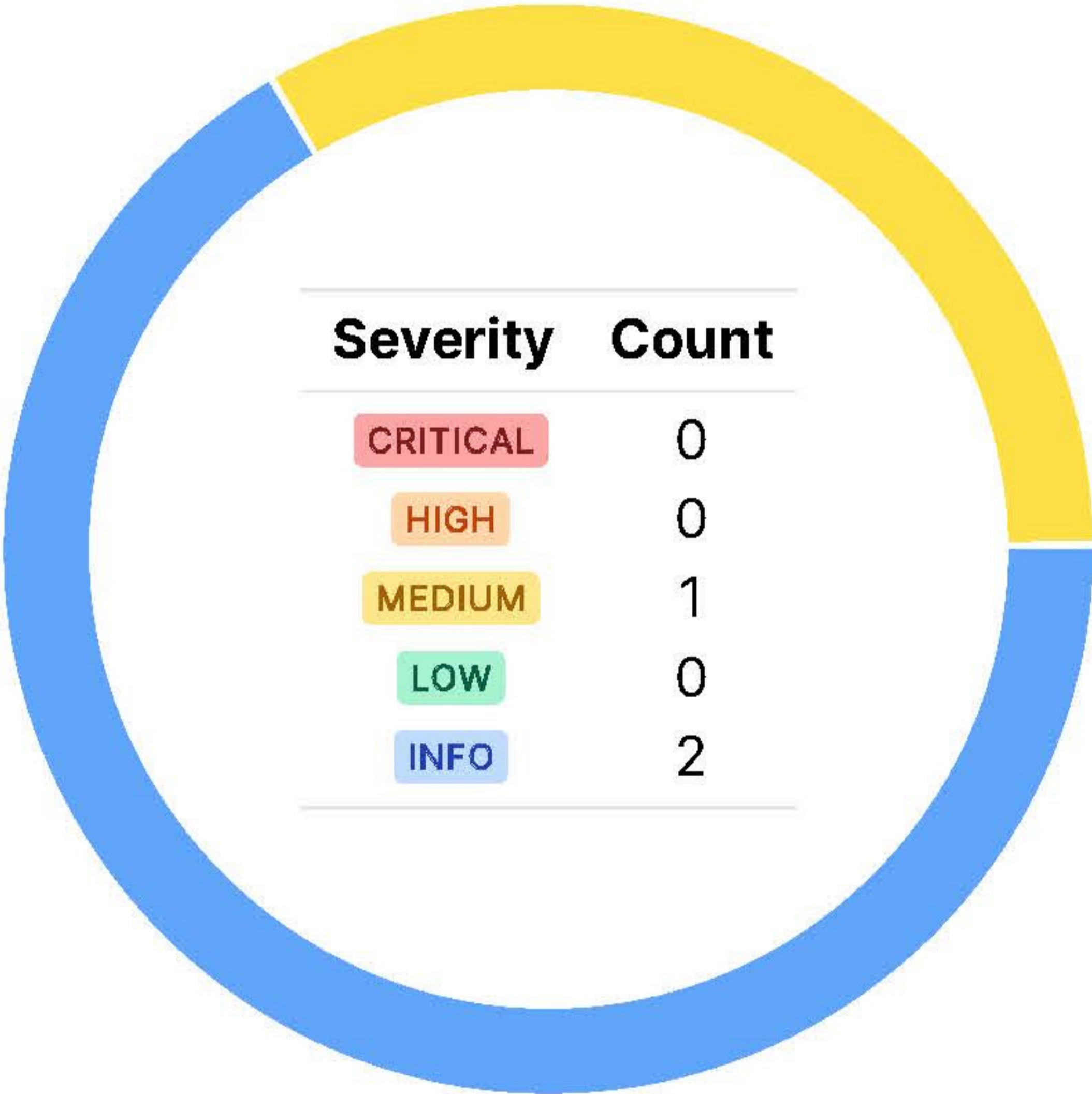
A brief description of the programs is as follows:

Name	Description
restaking	The Solana Liquid ETF Staking restaking program provides a mechanism for users to stake their LST tokens, earn rewards in the form of RST tokens, and manage their staked assets through freezing, thawing, staking, unstaking, minting, and burning operations.

02 — Findings

Overall, we reported 3 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.



03 — 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-SRP-ADV-00	MEDIUM	RESOLVED 	<code>freeze_rst_account</code> checks the frozen status of the <code>lst_ata</code> account instead of the <code>rst_ata</code> account.

Incorrect Account Check MEDIUM

OS-SRP-ADV-00

Description

The program utilizes the condition `if self.lst_ata.is_frozen()` in `restaking::freeze_rst_account` to check if the **RST** account (`rst_ata`) is already frozen. However, this check is incorrect as the program performs it on the `lst_ata` account.

```
>_ restaking-program/src/contexts/restaking.rs
```

rust

```
// Freeze RST token account if thawed
pub fn freeze_rst_account(&mut self) -> Result<()> {
    if self.lst_ata.is_frozen() {
        return Ok(())
    }

    let bump = [self.pool.bump];
    [...]
    freeze_account(ctx)
}
```

By checking the `is_frozen()` status of the incorrect account (`lst_ata` instead of `rst_ata`), the function may inaccurately determine that the **RST** account is already frozen when it is not. This may result in unintended behavior, such as skipping the freezing operation when freezing is necessary.

Remediation

Refactor `freeze_rst_account` to check the status of `rst_ata` account.

Patch

Fixed in [9401126](#) and [4a25804](#).

04 — 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-SRP-SUG-00	There is a possible centralization risk during token initialization if the supply of the <code>RST</code> token mint is a non-zero amount.
OS-SRP-SUG-01	Suggestion to include additional validations.

Centralization Risk

OS-SRP-SUG-00

Description

In `Initialize`, if the `rst_mint` account provided for initialization already possesses a non-zero supply of tokens, it introduces a potential vulnerability. The admin may mint additional tokens to the `rst_mint` account before transferring authority to control the minting process, artificially increasing the token supply.

```
>_ restaking-program/src/contexts/initialize.rs rust
pub fn initialize(&mut self, bumps: InitializeBumps) -> Result<()> {
    self.pool.set_inner(RestakingPool {
        lst_mint: self.lst_mint.key(),
        rst_mint: self.rst_mint.key(),
        bump: bumps.pool
    });
    ok(())
}
```

Consequently, the admin may transfer authority to control the minting process to another account while retaining a significant portion of the tokens. This allows the admin to manipulate the market by selling off the inflated tokens for profit, devaluing other users' assets.

Remediation

Ensure that the `rst_mint` account provided for initialization has a zero supply of tokens before proceeding with the initialization process. This pre-condition helps prevent the risk of token inflation.

Patch

Fixed in [9401126](#) by ensuring the supply of `rst_mint` in `initialize`.

Code Validation

OS-SRP-SUG-01

Description

Implement additional checks on the `token_account`, such as verifying the mint and account owner. However, skipping these checks should not have any security implications.

Remediation

Add the verification checks to the `token_account`.

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 the other 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.