TruFin Aptos Staker

Security Assessment

May 13th, 2024 — Prepared by OtterSec

Ajay Kunapareddy d1r3wolf@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

`TruFin` engaged OtterSec to assess the `Aptos Staker` aptos program. This assessment was conducted between April 30th and May 10th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 5 findings throughout this audit engagement.

In particular, we identified a vulnerability where the fee used for share price calculation is incorrect, resulting in users receiving extra rewards from others active in the subsequent epoch (OS-TRF-ADV-00). Additionally, we highlighted certain rounding issues that may deplete the protocol's assets, including the use of an incorrect amount to create an unlock request (OS-TRF-ADV-01) and the implementation of improper rounding directions within the unlock functionality (OS-TRF-ADV-02).

We also made some recommendations regarding minimum limitations on the staking amounts (OS-TRF-SUG-00) and the allocation system (OS-TRF-SUG-01).

Scope

The source code was delivered to us in a Git repository at https://github.com/TruFin-io/aptos-staker-ottersec. This audit was performed against commit 67005cc.

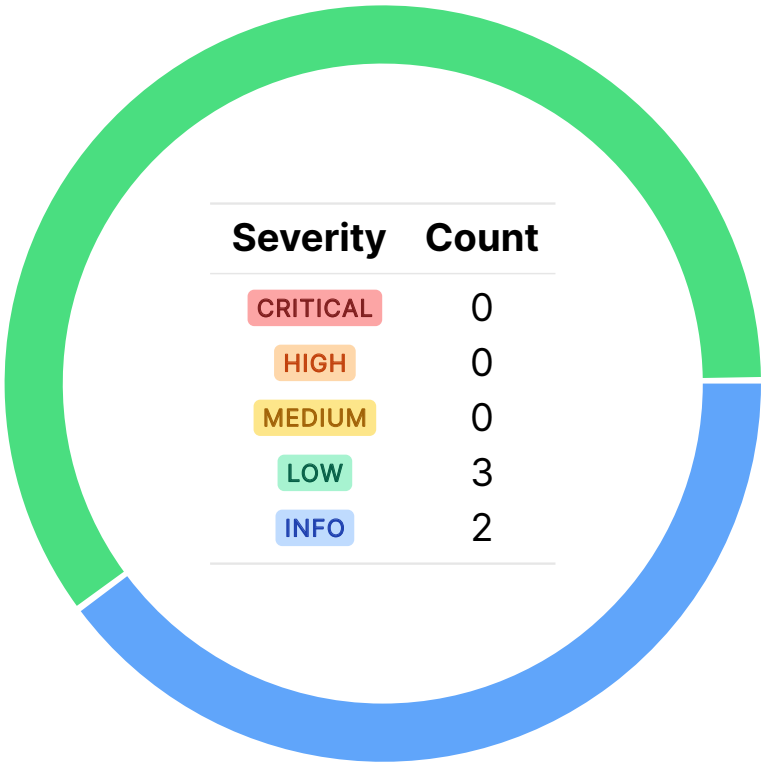
A brief description of the programs is as follows:

Name	Description
`Aptos Staker`	A liquid staking solution built on Aptos ecosystem.

02 — Findings

Overall, we reported 5 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	0
LOW	3
INFO	2

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-TRF-ADV-00	LOW	RESOLVED	`add_stake_fees` is utilized for share price calculation resulting in improper reward distribution.
OS-TRF-ADV-01	LOW	RESOLVED	The amount utilized to create an unlock request in `internal_unlock` is incorrect.
OS-TRF-ADV-02	LOW	RESOLVED	Improper rounding directions are utilized in `internal_unlock`.

Improper Reward Distribution LOW OS-TRF-ADV-00

Description

The `delegation_pool` transfers the `add_stake_fees` to the `NULL_SHAREHOLDER` and later returns them to the pool at the end of the epoch. This process occurs because the delegation pool does not distinguish between `active` and `pending_active` users, allowing both groups to receive rewards. To ensure `active` users get their rewards and `pending_active` users receive their `add_stake_fees` as rewards in the subsequent epoch, the system imposes `add_stake_fees` on users.

>_ *aptos-staker/sources/staker.move* rust

```
// calculate the delegation pool's "add stake fee" for this stake
let add_stake_fee = amount - (active_after_stake - active_before_stake);

// add the "add stake fee" amount to the staker's total amount of staking fees paid to this
    ↪ delegation pool for this epoch.
let pools_mut = borrow_global_mut<DelegationPools>(RESOURCE_ACCOUNT);
let pool_mut = smart_table::borrow_mut(&mut pools_mut.delegation_pools, delegation_pool);
pool_mut.add_stake_fees = pool_mut.add_stake_fees + add_stake_fee;
```

In this protocol, `add_stake_fees` which is distributed as rewards in the next epoch, is treated as stake within the same epoch, resulting in the minting of `TruAPT`. Consequently, users will receive extra rewards from others who are active in the subsequent epoch.

Remediation

Avoid considering `add_stake_fees` as staked amount and only mint `truAPT` for the amount that is actually being staked (`active_after_stake - active_before_stake`).

Patch

The development team acknowledges the issue and accepts the risk.

Inappropriate Unlock Amount LOW OS-TRF-ADV-01

Description

`internal_unlock` unlocks the `amount` from the delegation pool and burns the equivalent `truAPT` tokens. It creates an `UnlockRequest` for the user with the unstaked amount, which later facilitates the withdrawal of `APT` from the protocol.

>_ *aptos-staker/sources/staker.move* rust

```
    let unstaked_apt = pending_inactive - pending_inactive_pre_unlock;
    assert!(amount >= unstaked_apt && amount <= unstaked_apt + 2,
        ↪  error::invalid_argument(EINVALID_UNSTAKED_AMOUNT));

    let unlock_request = UnlockRequest{
        amount: amount,
        user: receiver,
        olc: olc,
        delegation_pool: pool,
        residual_rewards_collected: false
    };
```

`unstaked_apt` represents the actual stake amount being unstaked from the delegation pool. Additionally, `amount` sometimes exceeds the unstaked amount, potentially disbursing more `APT` than intended. Therefore, it is advisable to utilize `unstaked_apt` to create the unlock request.

Remediation

Utilize `unstaked_apt` instead of `amount` while creating the `UnlockRequest` in `internal_unlock`.

Patch

The development team acknowledges the issue and accepts the rounding costs to enhance user experience.

Incorrect Rounding Directions LOW OS-TRF-ADV-02

Description

`internal_unlock` unlocks the `amount` from the delegation pool and burns the equivalent `truAPT` tokens. It creates an `UnlockRequest` for the user with the unstaked amount, which later facilitates the withdrawal of `APT` from the protocol.

>_ *aptos-staker/sources/staker.move* rust

```
    let truAPT_amount;
    // if user's remaining balance falls below threshold, withdraw entire user stake
    if (max_withdraw - amount < MIN_COINS_ON_SHARES_POOL) {
        amount = max_withdraw;
        truAPT_amount = truAPT::balance_of(receiver);
    } else {
        truAPT_amount = convert_to_shares(amount);
    };
```

`amount` represents the stake requested for unstaking from the delegation pool and allocated to the user, and `truAPT_amount` denotes the tokens taken from the user and burned. The protocol benefits from rounding down when calculating `amount` and rounding up when calculating `truAPT_amount`.

Remediation

Employ `convert_to_assets(truAPT::balance_of(user))` to calculate the `amount` and apply rounding up when calculating the `truAPT_amount`.

Patch

The development team acknowledges the issue and accepts the rounding costs to enhance user experience.

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-TRF-SUG-00	The existing minimum limitation on staking may be slightly restrictive on users.
OS-TRF-SUG-01	Limitless allocations may result in an inaccurate total allocated amount.

Minimum Limit On Staking

OS-TRF-SUG-00

Description

Using `MIN_COINS_ON_SHARES_POOL` while staking and unlocking may be quite stringent. A more effective approach would be to deposit the required amount during the initialization phase. This would eliminate the need to constantly ensure that the pool maintains a sufficient stake, thereby simplifying the process and reducing potential complications.

Remediation

Deposit the required amount during the initialization phase to eliminate the need to constantly ensure that the pool maintains a sufficient stake.

Patch

Acknowledged by the development team.

Limitless Allocations OS-TRF-SUG-01

Description

The allocation model in the protocol used for reward distribution appears weak. While not a security vulnerability due to the centralized nature of these functions, it should be confirmed that these are intentional design choices.

For example:

1. A user may transfer `TruAPT` coins even after allocation.
2. A user may allocate the maximum withdrawal amount of `APT` multiple times: enabling users to excessively increase the allocated amount for a user, and in turn increase the total allocated amount.
3. A user may deallocate at any time without distributing rewards.

It is crucial to determine if these behaviors fall within the scope of the protocol's design or if they represent oversights that need to be addressed.

Remediation

Lock the amount of `truAPT` that is allocated so that it may not be transferred out of the account before distribution.

Patch

Acknowledged by the development team.

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.