Pyth Pull Oracle

Security Assessment

April 25th, 2024 — Prepared by OtterSec

Robert Chen notdeghost@osec.io

Harrison Green hgarrereyn@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Pyth Network engaged OtterSec to assess the `pyth-pull-oracle` program. This assessment was conducted between February 14th and February 19th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 4 findings throughout this audit engagement.

We made minor suggestions around rent exemption checks (OS-PPO-SUG-00) and stricter, consistent validation of guardian sets (OS-PPO-SUG-01). We also provided suggestions regarding ensuring adherence to coding best practices and code optimization (OS-PPO-SUG-03, OS-PPO-SUG-02).

Scope

The source code was delivered to us in a Git repository at https://github.com/pyth-network/pyth-crosschain. This audit was performed against commit 26a3ebb.

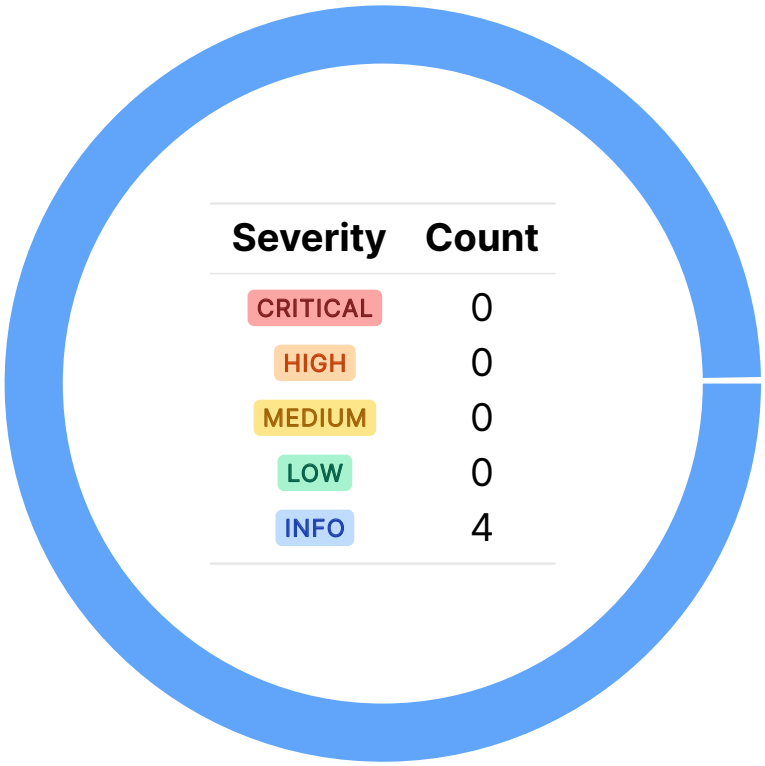
A brief description of the programs is as follows:

Name	Description
pyth-pull-oracle	The program is responsible for receiving and processing price updates from the Pyth network, validating them using cryptographic proofs and guardian signatures, and updating the program's state accordingly.

02 — 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	0
LOW	0
INFO	4

03 — 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-PPO-SUG-00	`post_price_update_from_vaa` incorrectly assumes that the payer account has no data during rent calculations.
OS-PPO-SUG-01	VAA posted via old guardian sets are accepted by `PostUpdate` instead of rejecting them.
OS-PPO-SUG-02	Recommendations regarding minor modifications to the code base.
OS-PPO-SUG-03	Suggestions regarding ensuring adherence to coding best practices and code optimization.

Inaccurate Rent Checks

OS-PPO-SUG-00

Description

The check for `InsufficientFunds` in `post_price_update_from_vaa` relies on `payer.lamports()` to determine the available balance, assuming the payer account has no associated data. However, this assumption may not always hold true, especially if the payer account already contains data.

Additionally, the function calculates the transaction fee amount and verifies if the treasury account has sufficient funds to cover it. However, it doesn't consider the scenario where the amount transferred to the treasury might not meet the rent-exemption threshold for the treasury account.

>_ pyth-solana-receiver/src/lib.rs rust

```rust
fn post_price_update_from_vaa<'info>(
    config: &Account<'info, Config>,
    payer: &Signer<'info>,
    write_authority: &Signer<'info>,
    treasury: &AccountInfo<'info>,
    price_update_account: &mut Account<'_, PriceUpdateV1>,
    vaa_components: &VaaComponents,
    vaa_payload: &[u8],
    price_update: &MerklePriceUpdate,
) -> Result<()> {
    if payer.lamports()
        < Rent::get()?
            .minimum_balance(0)
            .saturating_add(config.single_update_fee_in_lamports)
    {
        return err!(ReceiverError::InsufficientFunds);
    };
    [...]
}
```

Remediation

Utilize `payer.data_len()` instead of `payer.lamports()` to precisely assess the available balance. Additionally, transfer an extra fee amount to the `treasury` account to guarantee it meets the rent-xemption threshold.

Patch

Resolved in e0f94ee.

Acceptance Of Old Guardian Sets OS-PPO-SUG-01

Description

In `PostUpdate`, there is no explicit check to ensure the associated guardian set with the VAA is active or up-to-date. This means that even if the VAA is generated using an outdated or inactive guardian set, it will still be accepted and processed without verifying the current status of the guardian set.

However, `PostUpdateAtomic` correctly prevents the acceptance of VAA. generated using outdated or inactive guardian sets. The impact is minimal due to the timestamps in `PriceFeedMessage`.

>_ pyth-solana-receiver/src/lib.rs rust

```rust
pub fn post_update(ctx: Context<PostUpdate>, params: PostUpdateParams) -> Result<()> {
    [...]

    let vaa_components = VaaComponents {
        verification_level: VerificationLevel::Full,
        emitter_address:    encoded_vaa.try_emitter_address()?,
        emitter_chain:      encoded_vaa.try_emitter_chain()?,
    };

    post_price_update_from_vaa(
        config,
        payer,
        treasury,
        price_update_account,
        &vaa_components,
        encoded_vaa.try_payload()?.as_ref(),
        &params.merkle_price_update,
    )?;

    Ok(())
}
```

Remediation

Add a check for the guardian set's activity status in `PostUpdate`.

Patch

Resolved in 2ab72d9.

Code Refactoring OS-PPO-SUG-02

Description

1. There is a possibility of a governance transfer getting stuck in an intermediate state, where the target governance authority is set but the transfer is not finalized. Adding a `cancel_governance_authority_transfer` instruction to reset the target governance authority back to `None` will resolve this issue by allowing users to cancel the transfer if needed.
2. The verification level is determined by the number of signatures in the VAA, with a quorum threshold. However, it may be beneficial to automatically upgrade the verification level to `Full` if the number of signatures exceeds the quorum requirement. This will enhance security by ensuring that all signatures are fully verified, even if more signatures than required are present.

Remediation

1. Add a `cancel_governance_authority_transfer` instruction to reset the target governance authority back to `None`
2. Ensure the automatic upgrade of the verification level to `Full`, if the number of signatures exceeds the quorum requirement. Although this feature is currently infeasible due to implementation constraints, it should be considered for future updates.

Patch

Resolved in ddf0463.

Code Maturity OS-PPO-SUG-03

Description

1. In `deserialize_guardian_set_checked`, replace `try_deserialize_unchecked` with `try_deserialize` to deserialize the account data into a `GuardianSet` structure. The former skips the account discriminator check, potentially resulting in deserialization errors if the account data is not of the expected type. However, `AccountVariant` already performs its own discriminator check regardless of the method used.

>_ pyth-solana-receiver/src/lib.rs rust

```rust
fn deserialize_guardian_set_checked(
    account_info: &AccountInfo<'_>,
    wormhole: &Pubkey,
) -> Result<AccountVariant<GuardianSet>> {
    let mut guardian_set_data: &[u8] = &account_info.try_borrow_data()?;
    let guardian_set =
        AccountVariant::<GuardianSet>::try_deserialize_unchecked(&mut guardian_set_data)?;
    [...]
    Ok(guardian_set)
}
```

2. In `post_update_atomic`, consider moving the `minimum_signatures` check before the signature validation to save computing resources. It is common practice to verify the minimum required number of signatures or votes before validating individual signatures. This adjustment aligns with Wormhole's quorum check methodology and adheres to best practices.
3. In the `PostUpdate` structure, consider combining the constraints for the `treasury` account by merging the `mut` and `seeds` attributes. This adjustment enhances code readability and streamlines the declaration of the account.

Remediation

Implement the above-mentioned suggestions.

Patch

Resolved in e0f94ee and 7bf2782.

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.