Zellic

Pyth Network

Smart Contract Security Assessment

March 14, 2023

Prepared for:

Pyth Data Association

Prepared by:

Syed Faraz Abrar and Raj Agrawal

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Pyth Data Association from March 6th to March 8th, 2023. During this engagement, Zellic reviewed Pyth Network's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Can users set on-chain prices without passing in authentic price updates?
- Can an attacker take over the CosmWasm contract?
- Are there any error-prone migration or governance processes?
- Does the layout of the CosmWasm messages preserve forward compatibility for future extensions?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

During this assessment, the limited time frame of three days prevented us from fully looking at the test coverage to determine whether it could be improved.

1.3 Results

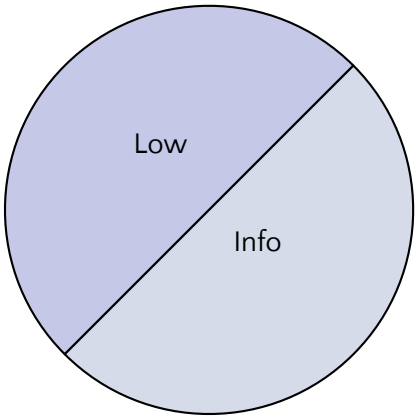
During our assessment on the scoped Pyth Network contracts, we discovered two findings. No critical issues were found. Of the two findings, one was of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Pyth

Data Association's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	1



2 Introduction

2.1 About Pyth Network

Pyth Network is an oracle providing real-time market data feeds to a number of different blockchains. The oracle creates price updates on the Pythnet blockchain and then sends them to other chains using the Wormhole network. These price updates can then be submitted and verified on various target chains. This audit encompasses the logic for receiving prices on CosmWasm blockchains.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the code base in general. We look for violations of industry best practices and guidelines and code quality stan-

dards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

Pyth Network Contracts

Repository	https://github.com/pyth-network/pyth-crosschain/
Version	: `6f700d17f0510460c3debb950c491524470a9a29`
Program	• CosmWasm Contracts
Type	CosmWasm
Platform	Cosmos

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of six person-days. The assessment was conducted over the course of three calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Syed Faraz Abrar, Engineer
faith@zellic.io

Raj Agrawal, Engineer
raj@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

March 3, 2023	Kick-off call
March 6, 2023	Start of primary review period
March 8, 2023	End of primary review period

3 Detailed Findings

3.1 New governance source may break transfer functionality

- **Target**: CosmWasm
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The `AuthorizeGovernanceDataSourceTransfer` action is used to modify the currently authorized governance source (i.e., the caller address that may perform governance actions through this contract). This is done through the `execute_governance_instruction()` function.

Specifically, the `AuthorizeGovernanceDataSourceTransfer` calls into `transfer_governance()`. This action allows the caller (who is the currently authorized governance source) to pass in a claim VAA that contains information about the new governance source to authorize. This claim VAA is supplied by the new governance source.

To prevent replay attacks, the claim VAA also contains a `governance_data_source_index`, which needs to be larger than the currently stored index. If it is not, it means that a previous `AuthorizeGovernanceDataSourceTransfer` message is being replayed, and thus the contract will reject it. This check can be seen in the `transfer_governance()` function:

```
fn transfer_governance(
    next_config: &mut ConfigInfo,
    current_config: &ConfigInfo,
    parsed_claim_vaa: &ParsedVAA,
) -> StdResult<Response> {
    // [ ... ]

    match claim_vaa_instruction.action {
        RequestGovernanceDataSourceTransfer {
            governance_data_source_index,
        } => {
            if current_config.governance_source_index
    >= governance_data_source_index {
                Err(PythContractError::OldGovernanceMessage)?
```

```
            }

            // [ ... ]
        }
        _ => Err(PythContractError::InvalidGgovernancePayload)?,
    }
}
```

The `governance_source_index` configuration property is a `u32`, so if the new governance source passes in a `RequestGovernanceDataSourceTransfer` action with the `governance_data_source_index` property set to the maximum `u32` value, then any subsequent `RequestGovernanceDataSourceTransfer` action can never have a higher `governance_data_source_index` property, and thus this action can never be performed again.

Impact

We do not consider this a security issue, as the new governance source is considered to be a trusted entity by the protocol already. We do however recommend that this be fixed, as the new governance source may accidentally brick this governance source transfer functionality of the contract by passing in the maximum `u32` value for the `governance_data_source_index`.

Recommendations

Consider adding a check such that the `governance_data_source_index` is higher than the currently stored `governance_source_index` but still within a certain amount.

Remediation

> Pyth Data Association acknowledges the finding and developed a patch for this issue: commit 3e104b41.

3.2 Open TODOs should be addressed

- **Target**: CosmWasm
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

During the course of the audit, we came across one open TODO.

Impact

The open TODO can be seen on line 610 of `contract.rs`:

```
// TODO: pass these in
```

Recommendations

Open TODOs should be addressed prior to final deployment of the contract.

Remediation

> Pyth Data Association acknowledges the finding and developed a patch for this issue: commit 3e104b41.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Implicit trust placed on parameters passed in by callers

Currently, the CosmWasm contract only accepts VAAs from certain whitelisted contracts on Pythnet. A trustless relayer can use signed VAAs from one of these contracts to execute price feed updates or governance instructions. The CosmWasm contract functions on the trust assumption that all data published by the emitter is already validated. The CosmWasm contract itself performs very minimal validation on the VAAs. Therefore, it is possible for the attestation contract to publish messages that could brick the contract in the following way (either due to a compromise or accidentally): Prevent the contract from accepting new price feed updates.

Below, we will dive a little deeper into how a caller may perform the above action, but it is crucial to note that Pyth Network has communicated with us on this issue already. They have stated that the caller is an attestation contract on Pythnet, and that the attestation contract itself enforces that the arguments passed into the CosmWasm contract calls are valid and cannot brick the contract.

Prevent new price feed updates

In the CosmWasm contract, the `update_price_feed_if_new()` function is used to check whether a submitted price feed update is considered new or old. New price feed updates have a `publish_time` property that is *after* the `publish_time` property of the previously accepted price feed update. Only new price feed updates can change the state of the contract, as old price feed updates are rejected.

The `publish_time` property itself is published by the oracle contract on Pythnet. This property has a type of `i64`. If the oracle contract publishes the maximum `i64` value for `publish_time` (either due to a bug or a compromise), then any subsequent price feed updates will be rejected, as the `publish_time` cannot go higher than the maximum `i64` value.

We have verified that the oracle contract simply uses the Solana clock's current timestamp for the `publish_time` property of the new price feed update. Therefore, this issue is currently mitigated, as the caller has no way to maliciously control the `publish_time` property of the submitted price feed update.

However, we find it important to note that the core issue here is that the CosmWasm

contract itself does not completely validate the arguments that are passed to it. Therefore, if the oracle or the attestation contract are compromised, the mitigations may not exist anymore, and thus the code will become open to exploitation. This would allow an attacker to submit a price feed update for a non-sensical price, or brick the CosmWasm contract by providing invalid arguments.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contract and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1 Module: contract.rs

Function: `execute_governance_instruction()`

Used to execute new governance instructions.

Inputs

- `vaa`
  - **Validation**: Checked to be a valid Wormhole message. Checked to be from an authorized governance source and chain. Checked to ensure that the governance action is valid. Checked to ensure the data source has supplied the necessary fee.
  - **Impact**: The price feed specified in the VAA is updated, with the new price set to the value in the VAA.

Branches and code coverage (including function calls)

Intended branches

- ☑ Valid `UpgradeContract` actions are handled successfully.
- ☑ Valid `AuthorizeGovernanceDataSourceTransfer` actions are handled successfully.
- ☑ Valid `SetDataSources` actions are handled successfully.
- ☑ Valid `SetFee` actions are handled successfully.
- ☑ Valid `SetValidPeriod` actions are handled successfully.

Negative behaviour

- ☑ VAA is rejected if the sequence is lower than the currently stored sequence.
- ☑ VAA is rejected if the emitter is not authorized.
- ☑ VAA is rejected if the emitter chain is not valid.

- ☐ VAA is rejected if it cannot be validated by wormhole (i.e., not enough wormhole guardian signatures).
- ☑ VAA is rejected if the target chain ID is invalid or not 0.
- ☑ VAA is rejected if deserialization fails for any reason.
- ☐ VAA is rejected if an invalid fee is provided for the `SetFee` action.
- ☐ VAA is rejected if a `RequestGovernanceDataSourceTransfer` action is specified as a direct governance instruction.
- ☑ VAA is rejected if the wrong module is specified for an action.
- ☑ VAA is rejected if a bad source index is specified for the `RequestGovernanceDataSourceTransfer` VAA.

Function: `update_price_feeds()`

Used to submit new price feed updates.

Inputs

- `vaa`
  - **Validation**: Checked to be a valid wormhole message. Checked to be from an authorized data source. Checked to be from a valid chain. Checked to ensure it's not a stale VAA. Checked to ensure the data source has supplied the necessary fee.
  - **Impact**: The price feed specified in the VAA is updated, with the new price set to the value in the VAA.

Branches and code coverage (including function calls)

Intended branches

- ☑ Valid VAAs lead to a successful price feed update
- ☑ Valid VAA with a stale or duplicate `publish_time` property does not cause a price update

Negative behaviour

- ☑ VAA is rejected if it does not contain the necessary fee.
- ☐ VAA is rejected if it cannot be validated by wormhole (i.e., not enough wormhole guardian signatures)
- ☑ VAA is rejected if the emitter is not an authorized data source.

6 Audit Results

At the time of our audit, the code was not deployed in production.

During our audit, we discovered two findings. Of these, one was low risk and one was a suggestion (informational). Pyth Data Association acknowledged all findings and implemented fixes.

6.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.