Zellic

Pyth Network

Smart Contract Patch Review

July 10, 2023

Prepared for:

Pyth Data Association

Prepared by:

Katerina Belotskaia and Filippo Cremonese

Zellic Inc.

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 to 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 email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Introduction

We conducted a review of a patch to the Pyth Network that introduced accumulators.

Accumulators are a new approach of generating verified prices for consumption in various networks. With accumulators, price updates are included in a Merkle tree, allowing to verify the authenticity of every individual price on chain without the need to verify every price included in the tree. This change increases Pyth scalability and brings gas savings and reliability benefits.

1.1 Scope

The engagement involved a review of the following targets:

Pyth Network

Repository	https://github.com/pyth-network/pyth-crosschain
Versions	https://github.com/pyth-network/pyth-crosschain/pull/929
Type	Solidity
Platform	EVM-compatible

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:

Katerina Belotskaia, Engineer
kate@zellic.io

Filippo Cremonese, Engineer
fcremo@zellic.io

1.2 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

additional code added to the assessed 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 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.

2 Smart Contract Patch Review

The patch involved updating the Pyth, PythGovernance, PythGovernanceInstructions, and ReceiverMessages contracts as well as the addition of the new PythAccumulator, UnsafeCalldataBytesLib, and MerkleTree contracts.

Added the PythAccumulator contract

- The contract includes the required functions to parse and verify new price data using the MerkleTree library.

Added the UnsafeCalldataBytesLib contract

- The BytesLib library has been revised to a version without safety checks to increase gas efficiency. The library contains functions that enable the concatenation, slicing, and type casting of bytes arrays.

Added the MerkleTree contract

- The new library was added to provide methods for the efficient verification and construction of Merkle tree proofs.

Updated the Pyth contract

- The `getUpdateFee` function has been updated to parse `updateData` to receive the total number of updates and calculate the fee using this number by calling `getTotalFee`, which multiplies the number of updates by `singleUpdateFeeInWei`.
- The `parsePriceFeedUpdates` function was updated to include the processing of accumulator update data.
- The new `getTotalFee` function was added to calculate the fee based on the number of updates.
- The new `findIndexOfPriceId` function was added to find the index of the given `targetPriceId` in the given `priceIds` array.
- The new `fillPriceFeedFromPriceInfo` function was added to fill the given `priceFeeds` array at index K with the given `priceId`, `priceInfo`, and `publishTime`. The function is called from the `parsePriceFeedUpdates` function.
- The `updatePriceInfosFromAccumulatorUpdate` function has been updated to verify and update prices using PythAccumulator.

Updated the PythGovernance contract

- The `executeGovernanceInstruction` has been updated to handle the new `SetWormholeAddress` operation.
- The new `setWormholeAddress` function was added to set the wormhole address

to a new value with sanity checks to ensure the new wormhole address can parse the payload correctly.

Updated the PythGovernanceInstructions contract

- The new `parseSetWormholeAddressPayload` function was added to parse the payload of the governance message to receive the new wormhole address.

Updated the ReceiverMessages contract

- The `parseAndVerifyVM` function has been revised to parse `encodedVM` data more efficiently in terms of gas consumption using the UnsafeCalldataBytesLib library.

3 Discussion

3.1 Hash truncation

We noticed that the `MerkleTree::hash` implementation truncates the output of the `keccak256` hash function, which is 32 bytes long, to only 20 bytes. This diminishes the collision resistance of the hash from 256 bits to 160 bits. The security level is still adequate, and we do not consider this a vulnerability. We believe the hash is truncated to save 12 bytes for each hash included in a Merkle proof.

3.2 Empty payload allowed `parseAndVerifyVM`

We note that `ReceiverMessages::parseAndVerifyVM` accepts empty payloads; it is our understanding that empty payloads are not intended to be used, so we suggest to consider rejecting empty payloads as a measure against potential mistakes. This could be accomplished by changing the following check:

```
uint hashIndex = index + (signersLen * 66);
if (hashIndex > encodedVM.length) {
if (hashIndex > encodedVM.length) {
    return (vm, false, "invalid signature length");
}
// Hash the body
vm.hash = keccak256(
    abi.encodePacked(
        keccak256(
            UnsafeCalldataBytesLib.sliceFrom(
                encodedVM,
                hashIndex
            )
        )
    )
);
```