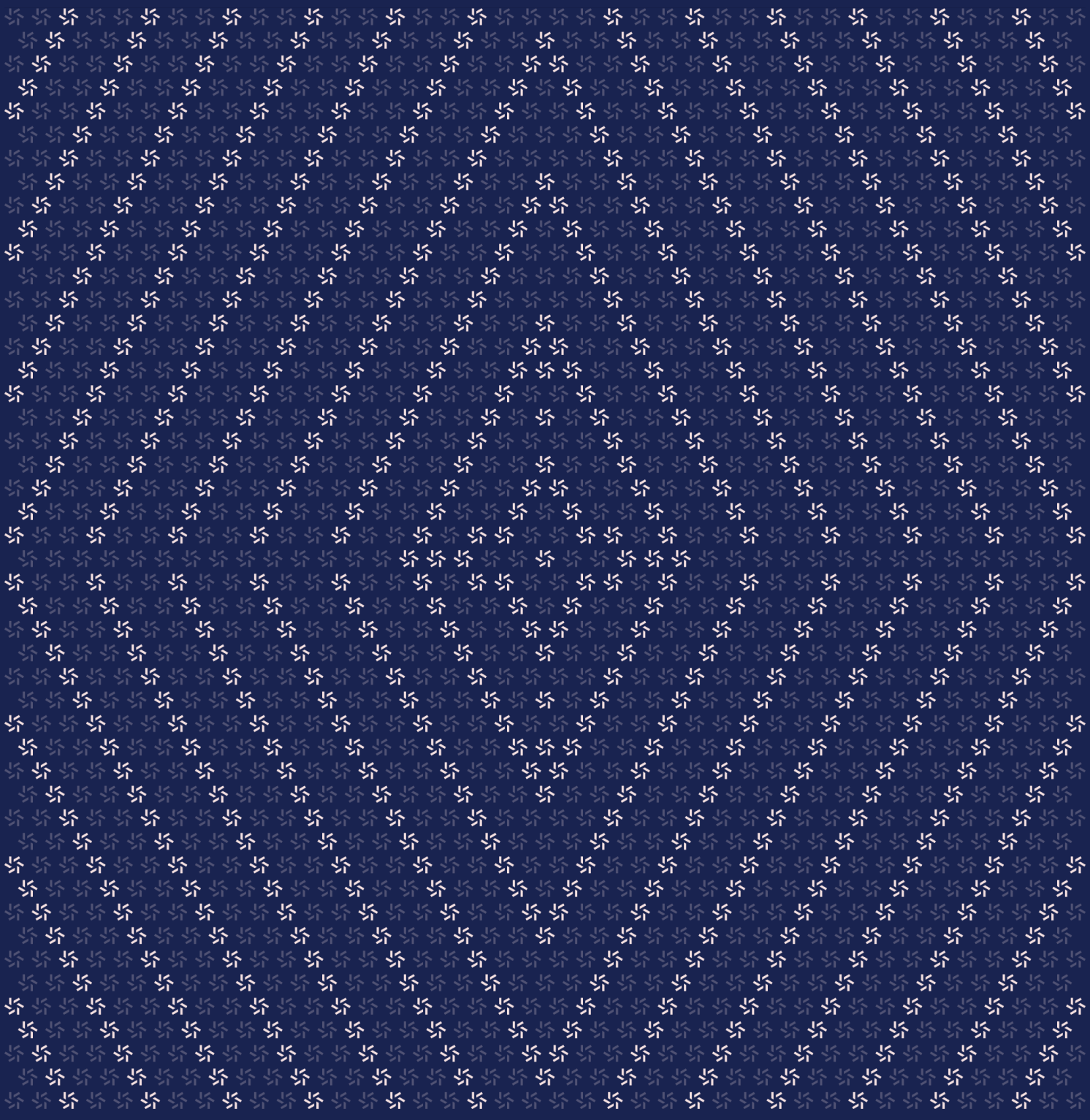
Zellic

Prepared for
John
Austin
Yei Finance

Prepared by
Hojung Han
Jaeeu Kim
Zellic

June 4, 2024

Oracle
Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Yei Finance on June 3rd, 2024. During this engagement, Zellic reviewed Oracle's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. 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:

- Could a malicious user manipulate the oracle contract to provide false data?
- Could a user get a wrong price from the oracle contract?

1.3. 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

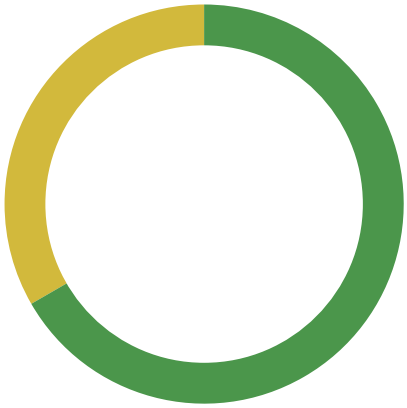
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Oracle contracts, we discovered three findings. No critical issues were found. One finding was of medium impact and two were of low impact.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	1
Low	2
Informational	0



2. Introduction

2.1. About Oracle

Yei Finance contributed the following description of Oracle:

> Yei Finance is a pioneering DeFi project built on Sei. We are the first money market on the Sei V2.

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 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 codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability 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:

Oracle Contracts

Repository	https://github.com/Yei-Finance/yei-oracle ↗
Version	yei-oracle: `60e663bef4d3bd37fd7ce31768e856f4f7e013dd`
Program	Oracle.sol
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of 1 person-days. The assessment was conducted over the course of 1 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:

Hojung Han
Engineer
hojung@zellic.io

Jaeeu Kim
Engineer
jaeeu@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

June 3, 2024	Start of primary review period
June 3, 2024	End of primary review period

3. Detailed Findings

3.1. Lack of stale price check in `getAssetPrice` function

Target	src/Oracle.sol		
Category	Protocol Risks	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The `latestAnswer` function in `Oracle.sol` retrieves price data from the Pyth network's contract using the `getPriceUnsafe` function. It is implemented as follows:

```
function latestAnswer() public view override returns (int256) {
    PythStructs.Price memory price = pyth.getPriceUnsafe(pythID);
    return int256(price.price);
}
```

Upon reviewing the use of the `latestAnswer` function, we found that, although it is not within the audit scope, it is utilized in the `getAssetPrice` function of the `AaveOracle.sol` contract.

This can be seen in the following GitHub link for AaveOracle.sol ↗.

The `getAssetPrice` function in the contract does not verify whether the price data returned by the oracle is stale. This oversight can lead to the use of outdated or incorrect price information, which may affect the contract's behavior and reliability.

```
function getAssetPrice(address asset) public view override returns (uint256) {
    AggregatorInterface source = assetsSources[asset];

    if (asset == BASE_CURRENCY) {
        return BASE_CURRENCY_UNIT;
    } else if (address(source) == address(0)) {
        return _fallbackOracle.getAssetPrice(asset);
    } else {
        int256 price = source.latestAnswer();
        if (price > 0) {
            return uint256(price);
        } else {
            return _fallbackOracle.getAssetPrice(asset);
        }
    }
}
```

Impact

Using stale price data can lead to incorrect financial calculations and decisions based on outdated information. This can have significant consequences, especially in financial applications where timely and accurate data is crucial.

In the past, instances of economic losses due to this issue have been reported, such as in this case with the Venus Protocol ↗.

Recommendations

Add code to check if the price data is within the last `n` minutes.

Remediation

This issue was fixed by Yei Finance Team in commit b7bc3ac846 ↗.

3.2. Unnecessary parameter usage in `getRoundData` function

Target	src/Oracle.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `getRoundData` function in the contract includes a parameter `_roundId` that is not utilized in the function's logic. Instead, the function retrieves the price data without considering the provided `_roundId`. This creates a potential for misuse, as users might expect the function to return data specific to the given `_roundId`, which is not the case here.

```
function getRoundData(
    uint80 _roundId      // [1] param
)
    external
    view
    returns (
        uint80 roundId,
        int256 answer,
        uint256 startedAt,
        uint256 updatedAt,
        uint80 answeredInRound
    )
{
    PythStructs.Price memory price = pyth.getPriceUnsafe(pythID); // [2] not
    using param
    return (
        _roundId, // [3] from param (directly)
        int256(price.price),
        price.publishTime,
        price.publishTime,
        _roundId
    );
}
```

Impact

The inclusion of an unused parameter can cause confusion and may lead to incorrect assumptions about the function's behavior. Users might expect the function to return data relevant to the provided `_roundId`, leading to potential misuse and unexpected outcomes.

Recommendations

Remove the unused `_roundId` parameter if it is not required for the function's logic.

Remediation

The Yei Finance Team has decided not to modify the code to maintain function signature compatibility with the Chainlink Oracle.

3.3. Centralization risk in `setPyth` function

Target	src/Oracle.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The following function can only be executed by the `owner` address.

In other words, if the private key of the `owner` address is leaked for some reason, such as an accident or a system compromise, it means that all price data returned by the Oracle can be manipulated.

```
function setPyth(address newPythAddress, bytes32 newPythID)
    external onlyOwner {
    pyth = IPyth(newPythAddress);
    pythID = newPythID;
}

[...]

function latestAnswer() public view override returns (int256) {
    PythStructs.Price memory price = pyth.getPriceUnsafe(pythID);
    return int256(price.price);
}
```

Impact

All types of dApps that obtain price data from this Oracle contract may suffer economic losses.

Recommendations

Reduce centralization risk where possible. One recommended method is to set the owner to a multi-sig wallet instead of an EOA.

Remediation

The Yei Finance team has decided to follow our recommendations.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts 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.

4.1. Module: Oracle.sol

Function: `getRoundData(uint80 _roundId)`

Inputs

- `_roundId`
  - **Validation**: N/A.
  - **Impact**: N/A.

Branches and code coverage

Intended branches

- ☑ Retrieve the latest price and the timestamp of the update.

Function call analysis

- `pyth.getPriceUnsafe(pythID)`
  - **External/Internal?** External.
  - **Argument control**: N/A.
  - **Impact**: Retrieve the latest price from the pyth-network contract.

Function: `latestAnswer()`

Branches and code coverage

Intended branches

- ☑ Retrieve the latest price in the pyth-network contract.

Function call analysis

- `pyth.getPriceUnsafe(pythID)`
  - **External/Internal?** External.

- **Argument control**: N/A.
- **Impact**: Retrieve the latest price data from the pyth-network contract.

Function: `updatePriceFeeds(bytes[] priceUpdateData)`

Inputs

- `priceUpdateData`
  - **Validation**: N/A.
  - **Impact**: A parameter used to update price data to the pyth-network contract.

Branches and code coverage

Intended branches

- ☑ Update the latest price data in the pyth-network contract.
- ☑ Refund any remaining ETH in the Oracle contract to `msg.sender`.

Negative behavior

- ☑ Revert if any problems occur in `msg.sender`'s receive or fallback functions while refunding the remaining ETH from the Oracle contract.

Function call analysis

- `pyth.updatePriceFeeds{value: fee}(priceUpdateData)`
  - **External/Internal?** External.
  - **Argument control**: `priceUpdateData`.
  - **Impact**: Update the latest price data in the pyth-network contract.

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Oracle contracts, we discovered three findings. No critical issues were found. One finding was of medium impact and two were of low impact. Yei Finance acknowledged all findings and implemented fixes.

5.1. 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 version reviewed during 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. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

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.