PUBLIC

Code Assessment

of the Sulu Extensions XX Smart Contracts

September 3, 2024

Produced for

by

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Enzyme Foundation with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Enzyme according to Scope to support you in forming an opinion on their security risks.

Enzyme Foundation implements two new Enzyme price feeds: a primitive price feed that converts ynETH (YieldNest ETH) to ETH and a derivative price feed that converts Stader SD tokens to ETH.

The most critical subjects covered in our audit are correct implementation of interfaces, external calls, as well as decimals usage. Security regarding all the aforementioned subjects is high.

The audit did not uncover any issues.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Enzyme repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	30 August 2024	651764b9322251049259d747b4d2f814752bd867	Initial Version

For the solidity smart contracts, the compiler version `0.8.19` was chosen.

The following files were included in the scope of the assessment:

`contracts/release/infrastructure/price-feeds/`:

- `primitives/ChainlinkLikeYnEthPriceFeed.sol`
- `derivatives/feeds/StaderSDPriceFeed.sol`

2.1.1 Excluded from scope

Anything not mentioned above is considered out of scope. In particular, the price feeds in scope make use of other price feeds which are assumed to function as intended. Attack vectors that involve trusted parties acting maliciously were not considered in this review.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Enzyme Foundation implements a new Chainlink-like ynETH price feed and a Stader SD token price feed for the Enzyme Sulu system.

2.2.1 Price Feed: ynETH

The provided price feed is a primitive price feed for YieldNest's liquid restaking token ynETH, returning the price of ynETH quoted in ETH (Wei). The price feed is wrapped in a Chainlink-like interface. It implements the `latestRoundData()` and `decimals()` functions from the `IChainlinkAggregator` interface.

The `latestRoundData()` function returns the current price computed for ynETH along with the `startedAt` and `updatedAt` values from the Chainlink stETH/ETH oracle. To compute the price of ynETH in ETH, the feed first gets the amount of ynETH per wstETH by using the `price_oracle()` for the ynETH/wstETH Twocrypto pool of Curve V2. It then retrieves the current rate of ETH per wstETH

using Enzyme's internal Chainlink-like wstETH/ETH primitive price feed. To avoid misinterpretation with an actual Chainlink price feed, the `roundId` and `answerInRound` return values are set to 0.

The `decimals()` function returns the precision of the price feed's `answer`, indicating the number of decimals for the quoted asset. That is 18 for the pricefeed.

Just like Chainlink price feeds, the `updatedAt` must be checked for staleness by the consumer whenever the price feed data is used. Note that, due to the use of two external asynchronous oracles, querying the price feed's `latestRoundData()` function multiple times may return varying `answer` rates for the same `updatedAt` value.

The contract stores decimals and units as constants and uses immutables for the addresses of external contracts.

2.2.2 Price Feed: Stader SD token

The `StaderSDPriceFeed` implements the `IDerivativePriceFeed` interface and only supports SD tokens as assets. It computes the value of the given amount of SD tokens in ETH (Wei) by using Stader's `getSDPriceInETH()` function. It is implicitly assumed that the decimal precision of the SD token is set to 18.

The contract stores units and addresses of external contracts as immutables.

2.2.3 Roles and Trust Model

Please refer to the main code assessment report and the extension reports for a general trust model of Sulu.

It is assumed that Chainlink prices update frequently enough to provide accurate data. The correctness of the ynETH price feed relies on the assumption that the Curve `price_oracle()` works as expected. This implies frequent usage, sufficient liquidity and reasonable parameter settings of the respective Curve pool. Specifically, it is required that multiple independent parties have significant liquidity in order to circumvent price manipulation attacks.

Additionally, it is assumed that the Stader TWAP Oracle behaves as expected. The parties updating the oracle are assumed to be trusted.

Due to the reliance on EMA and TWAP oracles, vault owners should be made aware that SD and ynETH tokens are considered to be risky assets.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

6.1 Curve Price Pool Precautions

Note Version 1

To calculate the price of `ynETH` in `ETH`, the Curve V2 `ynETH/wstETH` pool is queried. As the relative price of the assets is inferred by their liquidity in the pool, Enzyme Foundation and users should be aware of the following:

- The price of the pool is calculated using EMA. As EMA assigns some weight to previous price observations, the price reported will be different from the latest price and will only gradually converge to it as time goes by.
- EMA is used to make the price resistant to manipulation. An attacker can successfully manipulate the price if they manage to persist the manipulation within multiple blocks. Note that such an attack would open up arbitrage opportunities for other parties leading to losses for the attacker. Therefore, an attacker should be able to control where their transactions are placed in consecutive blocks. One simple example of multi-block control is the ability to place a transaction at the end of one and at the beginning of the next block. This is most realistic for the validator of the second block.
- The weight assigned to the current price compared to the previous ones is determined by the configuration of the pool. Different configurations can make the pool more or less resistant to price manipulation but also affect the rate at which the computed price converges to the latest price.
- The relative price of the assets depends on the liquidity (TVL) in the pool. Lower liquidity can allow for easier manipulation of the pool as the capital required to move the price in any direction is smaller.
- The price of the assets changes when their relative liquidity changes, e.g., when trades happen or when liquidity is added or removed. As a result, a pool with no activity will not be able to report a correct price. Note that there's a high correlation between TVL and activity.