PUBLIC

# Code Assessment

## of the Enzyme Onyx Smart Contracts

September 02, 2025

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 Onyx according to Scope to support you in forming an opinion on their security risks.

Enzyme Foundation implements Enzyme Onyx, a set of smart contracts to tokenize on- and off-chain value. It supports customizable deposit/redeem mechanisms, fees, and debt/credit tracking.

The most critical subjects covered in our audit are asset flow control, correctness of the fund valuation, fee handling, and precision of arithmetic operations. Security regarding all the aforementioned subjects is high.

The general subjects covered are upgradeability, documentation, and specification. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security as long as the admins follow the assumptions under Admin / owner assumptions.

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 | 1 |
| • Code Corrected | 1 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 1 |
| • Code Corrected | 1 |
| Low-Severity Findings | 5 |
| • Code Corrected | 3 |
| • Risk Accepted | 1 |
| • No Response | 1 |

# 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 Onyx 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 | 25 June 2025 | e74a1c0c760230222cf2412db8e8b230874bca2e | Initial Version |
| 2 | 14 July 2025 | a7c635b080ee06e96cb190567581a3055261c1ee | After Intermediate Report |
| 3 | 01 September 2025 | 6dbb223aca107aef73efdcc6875cfd4a2a8a95fe | License Update |

For the solidity smart contracts, the compiler version `0.8.28` was chosen.

The following files were in scope:

```
src/components/assets-sources/WithdrawableAssetsSource.sol
src/components/fees/FeeHandler.sol
src/components/fees/interfaces/IManagementFeeTracker.sol
src/components/fees/interfaces/IPerformanceFeeTracker.sol
src/components/fees/management-fee-trackers/ContinuousFlatRateManagementFeeTracker.sol
src/components/fees/performance-fee-trackers/ContinuousFlatRatePerformanceFeeTracker.sol
src/components/fees/utils/FeeTrackerHelpersMixin.sol
src/infra/oracles/OneToOneAggregator.sol
src/components/issuance/deposit-handlers/ERC7540LikeDepositQueue.sol
src/components/issuance/deposit-handlers/IERC7540LikeDepositHandler.sol
src/components/issuance/redeem-handlers/ERC7540LikeRedeemQueue.sol
src/components/issuance/redeem-handlers/IERC7540LikeRedeemHandler.sol
src/components/issuance/utils/ERC7540LikeIssuanceBase.sol
src/components/utils/ComponentHelpersMixin.sol
src/components/value/ValuationHandler.sol
src/components/value/position-trackers/IPositionTracker.sol
src/components/value/position-trackers/LinearCreditDebtTracker.sol
src/components/roles/LimitedAccessLimitedCallForwarder.sol
src/components/roles/OpenAccessLimitedCallForwarder.sol
src/factories/BeaconFactory.sol
src/interfaces/external/IChainlinkAggregator.sol
src/factories/ComponentBeaconFactory.sol
src/factories/ComponentBeaconProxy.sol
src/interfaces/IComponentProxy.sol
src/interfaces/IFeeHandler.sol
src/interfaces/ISharesTransferValidator.sol
src/interfaces/IValuationHandler.sol
src/global/Global.sol
```

```
src/global/utils/GlobalOwnable.sol
src/shares/Shares.sol
```

```
src/utils/Constants.sol
src/utils/StorageHelpersLib.sol
src/utils/ValueHelpersLib.sol
```

In Version 2, the following files have been added to the scope:

```
src/components/value/position-trackers/AccountERC20Tracker.sol
```

### 2.1.1 Excluded from scope

All other files not explicitly listed are out of scope of this review. This includes third-party libraries.

## 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 offers Enzyme Onyx, a set of smart contracts to tokenize on- and off-chain value.

Onyx's core functionalities include:

- The creation of an ERC20 tokenized representation of shares.
- Support for flexible deposit and redemption mechanisms, including asynchronous queues.
- Customizable fees.
- Customizable tracking of credits and debts.

The core of the system is located in the `Shares` contract. Other core contracts (e.g., fee handler, deposit queue, valuation handler) are treated as components of `Shares`.

### 2.2.1 Shares

Each Onyx instance has, at its core, a single `Shares` contract. This contract functions as an ERC20 token, representing the shares of the portfolio. Additionally, it is responsible for:

- Managing roles, storing the owner and administrators of the system. Admins and the owner have equal privileges with the exception of adding / removing admins.
- Managing components / tracking which components are authorized to interact with the system.

### 2.2.2 Components

The following contracts are components that interact with a single `Shares` instance.

#### 2.2.2.1 ValuationHandler

This contract is responsible for computing the price of the `Shares` ERC20 token.

The `ValuationHandler` allows admins to configure oracle settings for various asset types. These oracles will be utilized for conversions between the designated "value asset" (the asset the fund shares are priced in - typically the quote asset of the used oracle feeds) and different on-chain assets.

Furthermore, the contract allows to add position trackers used for on-chain accounting.

The codebase includes the `LinearCreditDebtTracker` which enables admins to add credit or debit items that are written off linearly over a certain duration of time (which can be 0 so that the items are instantly added to the sheet).

The main function of `ValuationHandler` is `updateShareValue()`. This function allows admins to report the equity of the fund, expressed in the value asset. Moreover, the function aggregates the reported equity with the sum of the values yield from the on-chain position trackers, obtaining the total value of the fund. After settling any fees, the total value is used to compute and store the value of a single share.

#### 2.2.2.2 FeeHandler

The system incorporates four distinct fee types:

- Entrance fee: a percentage of the assets deposited.
- Exit fee: a percentage of the assets withdrawn.
- Management fee: a percentage of the net value of the fund accrued over time.
- Performance fee: a percentage of the gains of the fund, determined by a fund-wide high watermark.

Entrance and exit fees are settled at the time of deposit and redemption time, respectively, while management and performance fees are settled exclusively when the share value is updated.

Note that the net value of the fund is computed by removing any settled fee from the fund's gross value.

The `FeeHandler` component is responsible for:

- Setting the fee percentage and recipient for each of the fee types.
- Settling and tracking fees within the system.
- Claiming the fees, i.e., sending the owed amount of `feeAsset` to the respective fee recipient.

Note that fees are tracked in the value asset but paid out in the `feeAsset` stored inside `FeeHandler`.

When settling management and performance fees, also referred to as dynamic fees, the value owed for each fee is computed by a specific fee tracker. Consequently, the system implements `ContinuousFlatRateManagementFeeTracker` for management fees and `ContinuousFlatRatePerformanceFeeTracker` for performance fees.

Management fees are calculated on the net value of the fund, while performance fees are calculated on the net value of the fund minus the management fees of each round.

Entry and exit fees can be configured with the 0-address as the recipient. In this case, fees are effectively distributed among the shareholders.

#### 2.2.2.3 Deposit and Redeem handlers

These components allow users to deposit assets into and withdraw assets from the fund. The codebase provides two asynchronous handlers: `ERC7540LikeDepositQueue` and `ERC7540LikeRedeemQueue`. Both components permit any user to request a deposit or redeem operation, storing each request under an increasing ID. The fund admin can then execute the requests in any order by specifying a set of request IDs to be executed.

Specifically, `ERC7540LikeDepositQueue` allows a user to deposit an amount of a given deposit asset, receiving a commensurate amount of shares based on the latest share price stored. The entrance fee is deducted from the shares minted to the depositor. Deposited assets are transferred to the `depositAssetsDest` address.

Similarly, `ERC7540LikeRedeemQueue` enables a user to burn an amount of shares, receiving a commensurate amount of redeem assets based on the latest share price stored. The exit fee is deducted from the assets sent to the redeemer. Withdrawn assets are pulled from the `redeemAssetsSrc` address.

Multiple deposit and redemption handlers can be added to the `Shares` contract, allowing for the deposit and redemption in different assets.

Note that the contracts do not meet the requirements of the ERC-7540 specification and aren't meant to - hence they are only ERC-7540 "alike".

#### 2.2.2.4 Roles Contracts

Enzyme Onyx implements two contracts for granular role management: `OpenAccessLimitedCallForwarder` and `LimitedAccessLimitedCallForwarder`. Both contracts are inteded to be set as admins in the `Shares` contract and expose the function `executeCalls()` which forwards calls to `Shares` or its components.

In more detail, `OpenAccessLimitedCallForwarder` allows anyone to call a restricted set of functions from a restricted set of contracts. For instance, `OpenAccessLimitedCallForwarder` could be used to allow any user to call the function `executeDepositRequests` of the contract `ERC7540LikeDepositQueue`.

The second contract, `LimitedAccessLimitedCallForwarder`, inherits from the open access limiter and further restricts the execution of `OpenAccessLimitedCallForwarder.executeCalls()` to a set of privileged users. This establishes the figure of "limited admins", which, unlike standard admins, can only perform a restricted set of actions.

#### 2.2.2.5 Wallets

The project defines a few different wallets in which funds can be stored for different purposes. While there are generally no restrictions on where the funds are held, assets must become accessible at some point, using the following wallets:

1. `depositAssetsDest`: Address to which assets are sent from the deposit queue after deposit requests have been executed. These assets are then available for investment.
2. `feeAssetsSrc`: Address from which fees can be claimed. While fees accrue during each valuation step, they can only be claimed when assets are sent to this address. Fees remain part of the fund valuation until they are claimed.
3. `redeemAssetsSrc`: Address from which redemptions are sent to the respective users. For successful execution of redemption requests, enough assets must be held in this wallet.

#### 2.2.2.6 Auxiliary contracts

Some auxiliary contracts can be used in conjunction with the main contracts:

1. **WithdrawableAssetsSource**: A minimal wallet contract that can be used to accept assets and give approvals to the `Shares` contract.
2. **OneToOneAggregator**: A dummy contract mimicking a Chainlink oracle to always return a price of one.

### 2.2.3 Deployment

Enzyme Foundation will utilize beacon proxies for the deployment of the contracts.

In particular, the `Shares` contract will be deployed behind a `BeaconProxy` from the `BeaconFactory`.

Components will be deployed behind a `ComponentBeaconProxy` from the `ComponentBeaconFactory`.

The beacon proxy pattern is employed to facilitate simultaneous upgrades across different Onyx instances by deploying a new implementation and updating the implementation address returned by the beacons.

The owner of the beacon factories is the owner of the `Global` contract, which must be deployed first. This is the only contract owned by Enzyme Foundation.

### 2.2.4 Changes in Version 2

In Version 2 of the protocol, the following changes have been added:

1. Fees are only claimable by the admins and the owner.
2. Storage locations are now hardcoded but checked against the respective string in the constructor.
3. `depositAssetsDest`, `feeAssetsSrc`, and `redeemAssetsSrc` have been removed in favor of the `Shares` contract being the sole holder of tokens.
4. A new contract `AccountERC20Tracker` has been added that allows to track the value of a single address's tokens.
5. Instead of Chainlink oracles, assets are now priced based on asset prices set directly by the admins or the owner.

## 2.3 Trust Model

**Global Owner** The owner of the `Global` contract and, consequently, of the beacon factories.

- Fully trusted.
- Has full control of the factories
- Could fully drain the system.

**Fund Owner** The owner of the Onyx instance, has full control of the fund and can designate admins.

- Fully trusted.
- Can add and remove admins.
- Has full control over the fund.
- Could drain the system.

**Fund Admins** Admins have the same privileges as the fund owner with the exception that they cannot modify the admin list.

- Fully trusted.
- Have full control over the fund.
- Can manage the set and capabilities of limited admins.
- Could drain the system.

We assume the admins to correctly configure the fund and its components. In particular, admins will exclusively add trusted components to the Shares instance.

Moreover, we assume that the fund admins will properly configure the oracles. Specifically, since the value asset used by the fund is not enforced in the code, it is the admins' responsibility to ensure that the price oracles convert to and from the correct value asset. Since Version 2, admins (and owners) are instead trusted to set the correct asset prices at the right time without opening up frontrunning possibilities (i.e., updating the valuation while asset prices in the contract are still outdated).

Apart from these trust assumptions, we have detailed a list of further assumptions on the behavior of the admins to guarantee a safe operation of the fund: Admin / owner assumptions.

The assumptions above are equally valid for the fund owner.

**Limited Admins** Limited admins interact with the `LimitedAccessLimitedCallForwarder` contract to perform a restricted set of actions. The list of limited admins and the set of permitted actions is

managed by the fund admins. Although the capabilities of limited admins can vary, they should be trusted to behave honestly with respect to the privileges they are granted.

- Fully trusted in the scope they have been assigned to. E.g., if a limited admin has permission to update the share valuation, then they are trusted to update the valuation correctly.
- Could drain the system if allowed by their privileges.

Furthermore, limited admins should make sure not to send multiple calls with value in the `LimitedAccessLimitedCallForwarder` that do not sum up to the `msg.value`.

**Fee recipients** For each type of fee, a different recipient can be configured. These recipients can claim fees via `FeeHandler.claimFees()`.

- Minimally trusted as they can influence the valuation by frontrunning the call to `ValuationHandler.updateShareValue()`.

**Users** Users of the fund can create deposit and redemption requests.

- Untrusted.
- Can deploy contracts directly from the `BeaconFactory` and `ComponentBeaconFactory`. Such deployments are therefore trusted.

Users can theoretically become trusted if the `OpenAccessLimitedCallForwarder` is used. We therefore assume the contract is used only when no abuse potential exists.

Users should make sure to not send multiple calls with value in the `OpenAccessLimitedCallForwarder` that do not sum up to the `msg.value`.

**Price Oracles** The system heavily relies on price oracles to compute the share value and perform conversions between assets.

- Trusted to return correct prices and correct price decimals.
- Generally, the price oracles are assumed to be either official Chainlink price feeds or the `OneToOneAggregator` and are compatible with the value asset of the associated fund.
- The `OneToOneAggregator` is assumed to only be used when there is no risk of a depeg of the given asset.

**Tokens** It is assumed that the project will not be used with the following ERC-20 token types:

- Rebasing tokens: Increasing token amounts cannot be accounted for by the system.
- Fee-taking tokens: Decreasing token amounts cannot be accounted for by the system.
- Tokens capping the transfer amount to the user's balance: Discrepancies between user balances and the actual amount of tokens deposited would emerge.
- Tokens with callbacks to the recipient: Deposit / redemption request executions could be blocked (although this can be mitigated by excluding blocking users).

# 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 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

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 | 2 |



## 5.1 Deposits Can Be Stolen By Inflating The Share Price RISKACCEPT

Security Low Version 2

*CS-ONYX-011*

In Version 2, Enzyme Foundation added the new `AccountERC20Tracker` position tracker, which tracks the balance of arbitrary ERC-20 assets held by a wallet `_account`. The tracker relies on `asset.balanceOf(_account)` to determine the value of the position. As a consequence, donations to `_account` influence the value of the position. Importantly, an adversary might be able to inflate the share price by performing a donation, managing to partially steal a user's deposit.

The following scenario provides an example of the attack.

Assume the value asset to be USD and the deposit and redeem asset to be DAI. Also, for simplicity, assume no entry, exit or dynamic fees.

1. The deployer deploys a new fund.
2. An attacker makes a deposit request for 1 wei DAI.
3. An admin updates the valuation, setting the `valuePerShare` to the default of 1e18.
4. An admin executes the attacker's deposit request.
5. The attacker receives 1 wei of shares.
6. A victim makes a deposit request for 100 wei DAI.
7. An admin updates the valuation. The transaction is frontrun by the attacker, donating 50 wei DAI to the account tracked by the `AccountERC20Tracker`. `valuePerShare` now is 51e18.
8. An admin executes the victim's deposit request.
9. The victim receives 1 wei shares.

10. The attacker now redeems their 1 wei shares.
11. An admin executes the attacker's redemption request. The attacker receives 75 wei DAI, stealing 24 wei DAI from the victim.

Note that an attacker can always frontrun the valuation calculation but for this attack to succeed, admins must execute a single deposit with very low value.

---

**Risk Accepted:**

Enzyme Foundation has acknowledged the possible risks stating:

> Onyx intentionally does not protect against inflation/donation/low shares supply attack vectors in its core. It is recommended that admins with untrusted shareholders mint existential shares to eliminate any such attacks.

## 5.2 Fee Change After Requests

Design | Low | Version 1 | Risk Accepted

*CS-ONYX-003*

Entry and exit fees can be changed by the admins of the contracts at any time. When users want to deposit or redeem funds, they have to first create a request that is later executed by the admins. When creating a request, no fees are taken. Instead, the fees are taken during the request execution (at a later point in time). Additionally, the requests don't persist the fee values that were in effect during the creation of the request.

It is therefore possible that a user creates a request and then an admin increases the associated fee and executes their request without the user being able to intervene.

---

**Risk Accepted:**

Enzyme Foundation has acknowledged the issue stating the following:

> Admins are trusted and can adopt their own practices for whether or not to process requests that were submitted prior to an important fee change

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

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

| | |
|---|---|
| Critical-Severity Findings | 1 |

- Unsafe Downcast in Fee Claiming Allows Theft of Funds and Permanent DoS Code Corrected

| | |
|---|---|
| High-Severity Findings | 0 |
| Medium-Severity Findings | 1 |

- Unburned Exit Fee Shares Permanently Inflate Total Supply and Deflate Share Price Code Corrected

| | |
|---|---|
| Low-Severity Findings | 3 |

- Performance Fee High-Water Mark (HWM) Misalignment Code Corrected
- Rounding Precision Error Code Corrected
- Valuation Race Condition Code Corrected

| | |
|---|---|
| Informational Findings | 2 |

- Inconsistent Error Handling Code Corrected
- Mismatched Namespace ID in NatSpec Annotation Code Corrected

## 6.1 Unsafe Downcast in Fee Claiming Allows Theft of Funds and Permanent DoS

Security Critical Version 1 Code Corrected

*CS-ONYX-001*

The `FeeHandler.claimFees()` function allows a user to claim fees, but it fails to validate that the claimed amount `_value` is not greater than the fees actually owed to the user.

When a user claims a `_value` even slightly larger than their balance, the internal `__updateValueOwed()` function calculates the new balance by subtracting a larger number from a smaller one. This results in a negative `int256` value, which, when unsafely cast to `uint256`, underflows.

This single vulnerability has multiple critical impacts:

1. **Theft of Funds:** Anyone can claim more fees than they are owed (zero in the case of unintended fee recipients), draining assets from the `feeAssetsSrc` contract. Since no revert takes place on underflow, users can claim any amount of tokens owned by the fee source address.
2. **Permanent Denial of Service:** The underflow can also corrupt the `totalFeesOwed` state variable. This will cause all future attempts by the admin to call `ValuationHandler.updateShareValue()` to revert, because its internal calculation `netValue = _totalPositionsValue - getTotalValueOwed()` will always revert on underflow.

Additionally, the delta value `-int256(_value)` is passed to `__updateValueOwed()` from a `uint256` value which can equally result in an overflow. This is currently not exploitable though as the value is also

transformed with `ValuationHandler.convertValueToAssetAmount()` which will always result in a revert for numbers big enough to cause an overflow.

---

**Code Corrected:**

Enzyme Foundation has corrected the issue by splitting the `__updateValueOwed()` function into two separate functions: `__increaseValueOwed()` which increases the value owed to a user and the total value owed, and `_decreaseValueOwed` which accordingly decreases both variables. Both functions exclusively work with `uint256`, therefore removing the need for casts to and from `int256`.

## 6.2 Unburned Exit Fee Shares Permanently Inflate Total Supply and Deflate Share Price

Correctness | Medium | Version 1 | Code Corrected

*CS-ONYX-002*

An accounting bug in the redemption process causes exit fee shares to never be burned. They remain in the `ERC7540LikeRedeemQueue` contract's balance while still being counted in the `Shares` contract's `totalSupply()`. This permanently inflates the total supply, which is the denominator of all share price calculations. As a result, every redemption with a non-zero exit fee deflates the share price for all remaining investors, causing a direct and accumulating loss of value. When a non-privileged user initiates a redemption, their shares are transferred to the `ERC7540LikeRedeemQueue` contract. During execution, the system calculates the exit fee in shares (`feeSharesAmount`) but only burns the `netShares`.

---

**Code Corrected:**

Enzyme Foundation has corrected the code by burning the gross shares (`request.sharesAmount`) when executing redemption requests.

## 6.3 Performance Fee High-Water Mark (HWM) Misalignment

Design | Low | Version 1 | Code Corrected

*CS-ONYX-005*

The performance fee calculation has a subtle flaw in how the high-water mark (HWM) is updated. The process is as follows:

1. The `ValuationHandler` calls the `FeeHandler` to settle dynamic fees.
2. The `FeeHandler` passes a `_netValue` (total value minus previous fees and current management fee) to the `ContinuousFlatRatePerformanceFeeTracker`.
3. The `settlePerformanceFee()` function calculates `valuePerShare` based on this `_netValue` and uses it to determine the performance fee due.
4. Crucially, it then updates the HWM to this `valuePerShare`.
5. Afterwards, the `ValuationHandler` calculates the final share value for investors by subtracting all fees, including the just-calculated performance fee.

This creates a misalignment: The HWM is set based on a pre-performance-fee valuation, while the actual share valuation is post-performance-fee. The HWM is thus always set higher than the value per share at the end of the period. This means the fund must not only generate a profit in the next period but also make up for the value of the last performance fee before new performance fees can be charged.

---

**Code Corrected:**

Enzyme Foundation has corrected the code by changing how the HWM is updated. The function `settlePerformanceFee()` subtracts the performance fees from `_netValue`, obtaining `netValueIncludingFee`. The HWM is then set to the value per share computed using `netValueIncludingFee`.

## 6.4 Rounding Precision Error

**Correctness** **Low** **Version 1** **Code Corrected**

*CS-ONYX-006*

`ValueHelpersLib.convert()` can be imprecise when the quote asset has low decimals. This is exacerbated by the fact that asset conversions are only performed for 1 share. Consider the following example:

1. A user redeems 10 000 shares from a redeem queue with GUSD as the asset.
2. The oracle price of GUSD has 8 decimals.
3. The current share price is `1.019 * 10**18`.
4. `sharePriceInRedeemAsset` is calculated as follows:

$$\frac{1.019 * 10^{18} * 10^{8} * 10^{2}}{10^{8} * 10^{18}} = 101.9$$

1. The user will receive 10 100 GUSD.
2. Performing the same calculation with all withdrawn shares results in 10 190 GUSD.

---

**Code Corrected:**

Enzyme Foundation has corrected the code by changing the logic inside `executeDepositRequests()` and `executeRedeemRequests()`. The conversion is now done without intermediate scaling down the share price to the deposit / redemption asset, leaving enough precision for only miniscule rounding errors.

## 6.5 Valuation Race Condition

**Design** **Low** **Version 1** **Code Corrected**

*CS-ONYX-007*

`ValuationHandler.updateShareValue()` calls the `FeeHandler` to get the amount of fees currently owed to fee receivers. This amount is then subtracted from the total positions value to get the total fund value with which the value of the fund's shares can be calculated.

`FeeHandler.getTotalValueOwed()` returns all fees that have been accrued since the start of the fund and have not been claimed yet. This means that, for accurate calculation, admins must pass a total fund value that includes fees only as long as they haven't been claimed yet (either directly through `_untrackedPositionsValue` or indirectly through items in an associated positions tracker).

When an admin submits such a position value containing some unclaimed fees on-chain and at the same time, a fee recipient (which might be a different entity) claims fees, a race condition can occur when the fee recipient's transaction is executed before the transaction of the admin. In this case, the total fees owed would be reduced while the positions value submitted by the admin stays the same, resulting in an inflated total value of the fund.

---

**Code Corrected:**

Enzyme Foundation has corrected the code by only allowing admins to claim fees. Since both price update and fee claiming are performed by the admins, race conditions are avoided.

## 6.6 Inconsistent Error Handling

Informational | Version 1 | Code Corrected

*CS-ONYX-009*

The contract `ValuationHandler` exhibits inconsistent behavior in some functions. For example, `ValuationHandler.removePositionTracker()` reverts if the tracker being removed does not exist. However, `ValuationHandler.unsetAssetOracle()` simply uses `delete` on the mapping entry and does not revert if an oracle for the given asset was not set. This inconsistency can lead to developer confusion and potentially hide state-related bugs in off-chain software interacting with the contract.

---

**Code Corrected:**

Enzyme Foundation has removed the `unsetAssetOracle()` function when removing ChainLink oracles in favour of manualy-set rates.

## 6.7 Mismatched Namespace ID in NatSpec Annotation

Informational | Version 1 | Code Corrected

*CS-ONYX-004*

`ContinuousFlatRateManagementFeeTracker` and `ContinuousFlatRatePerformanceFeeTracker` violate a requirement of EIP-7201. The EIP requires:

> A namespace in a contract should be implemented as a struct type. These structs should be annotated with the NatSpec tag `@custom:storage-location <FORMULA_ID>:<NAMESPACE_ID>`, where `<NAMESPACE_ID>` identifies a formula used to compute the storage location where the namespace is rooted, based on the namespace id.

The two contracts have a mismatch between the namespace identifier used in the code to derive the storage location and the identifier documented in the `@custom:storage-location` NatSpec tag. The code computes the namespace ID by prefixing the given string with `enzyme.` (e.g., `enzyme.ManagementFeeTracker`), but the NatSpec annotation incorrectly includes an extra `.storage` segment (e.g., `enzyme.storage.ManagementFeeTracker`). This can mislead developers and cause tooling that relies on this annotation to fail to locate storage variables correctly.

---

**Code Corrected:**

Enzyme Foundation has corrected the code by removing the extra `.storage` segments.

# 7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

## 7.1 Denial of Service Due to Rounding Error

**Informational** **Version 1** **Acknowledged**

*CS-ONYX-008*

The `executeDepositRequests()` function is vulnerable to a (minor) Denial of Service attack. The function processes an array of request IDs in a batch. If a request included in the batch is submitted with an asset amount of 1 wei and the share value is greater than 1e18, the function calculates zero shares, triggering a `require(netShares > 0)` check that reverts the entire transaction. This same vulnerability exists in `ERC7540LikeRedeemQueue` if the share price is below 1e18.

---

**Acknowledged:**

Enzyme Foundation has acknowledged with the following statement:

> Actively filtering-out requests that are too small is part of the intended admin process, so there is no DoS

## 7.2 Missing Event Indexes

**Informational** **Version 1** **Acknowledged**

*CS-ONYX-010*

With the exception of the events in `IERC7540LikeDepositHandler` and `IERC7540LikeRedeemHandler`, none of the events emitted by Enzyme Onyx contracts have indexes set up for any of their parameters. It is recommended to index the relevant event parameters to allow integrators and dApps to quickly search for these and thus simplify off-chain computation.

---

**Acknowledged:**

Enzyme Foundation has decided not to index events, as they do not rely on indexed events for off-chain computation.

# 8 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.

## 8.1 Admin / Owner Assumptions

Note Version 1

The protocol's design allows fund owners and their chosen admins a great amount of flexibility. There are, however, certain cases where this flexibility can result in problematic behavior. The following list examines these cases:

1. Valuation updates should only be performed with the correct values. Otherwise, the fund's share price will be inflated / deflated, affecting user deposits and redemptions.
2. Executions of deposit / redemption requests should only happen directly (i.e., atomically, to prevent front-running) after the valuation has been updated. Otherwise, the share price will not reflect the actual value of the fund at the time of execution, allowing users to either benefit or lose unfairly.
3. Changing the asset on any of the contracts also requires setting up a corresponding oracle in the `ValuationHandler` beforehand. Otherwise, the contracts will no longer be usable. In Version 2, admins are responsible for manually setting the correct rate of an asset.
4. `ContinuousFlatRateManagementFeeTracker.resetLastSettled()` should only be called right before the fund is started. Setting the settlement date earlier allows to unfairly accumulate additional management fees during the first settlement.
5. `ContinuousFlatRatePerformanceFeeTracker.resetHighWaterMark()` should not be called in a way that disadvantages users. For example, if `1e18` is deposited after deployment and then `1` wei is reported as total value, the share price is set to 1, allowing the fund to unfairly accumulate additional performance fees during the first settlement.
6. Valuation updates with a positive total value should not be performed on a fund with no shares. Otherwise, management fees are accrued unfairly for subsequent depositors.
7. `minRequestDuration` should always be set to a positive value in `ERC7540LikeDepositQueue` and `ERC7540LikeRedeemQueue`. Otherwise, request executions might unexpectedly fail due to the controllers of the executed requests cancelling them before they can be executed.
8. When the `LinearCreditDebtTracker` is used, a maximum number of items should not be exceeded to prevent DoS due to out-of-gas. Items can be aggregated to prevent this.
9. Funds should not be misappropriated.
10. Admins should not execute deposit / redemption requests of very low value. Otherwise, the transaction could revert. Additionally, in new funds, this could lead to an inflation attack vector.
11. In Version 2, admins should claim the fees on behalf of the fee recipient(s) appropriately.

## 8.2 Inflexible Performance Fee

Note Version 1

Performance fees are calculated based on a high watermark that is updated each time the performance fee is settled. The high watermark is the highest value of the fund's shares at the time of the last performance fee settlement.

It is stored on the fund-level instead of the individual user-level. This can lead to an advantage for late joiners in the following scenario:

1. The valuation of a fund starts at 1.
2. Over time, the valuation increases to 2.
3. Users that deposited in the beginning paid performance fees on the full gain from 1 up to 2.
4. Now the fund incurs losses of 50%, bringing its valuation back to 1.
5. New users can now deposit and will not pay performance fees until the fund reaches a valuation of 2 again, even though they will experience 100% gains along the way.

## 8.3 Management Fee Compound Depreciation

Note Version 1

`ContinuousFlatRateManagementFeeTracker` settles the management fee of a fund each time its valuation is updated. The fees are calculated as a percentage of the value of the fund minus the already accrued fees. This results in compound depreciation behavior - the amount of fees decreases with the frequency at which the valuation is updated.

For example, if the total value (1000 USD) of a fund stays the same throughout one year (i.e., the funds are not invested), calculating a management fee of 10% after 1 year results in a fee of 100 USD. However, if the same fee is calculated monthly, it amounts to only 95.54 USD.

## 8.4 Unexpected Behavior Due to Third Party Fee Funds

Note Version 1

`ValuationHandler.updateShareValue()` calls the `FeeHandler` to get the amount of fees currently owed to fee receivers. This amount is then subtracted from the total positions value to get the total fund value with which the value of the fund's shares can be calculated.

Fees have to be actively sent to a fee source address from which they can then be claimed. If no funds exist at the address, no fees can be claimed even though they might be owed, allowing the fund owner some flexibility.

Fund owners should be aware that third parties could theoretically send funds to the fee source address which would allow claiming fees. In such a case, the total fees owed would be reduced without the funds knowledge, leading to an inflated valuation in the next valuation update if the address and / or the `FeeHandler` are not monitored properly and the fund owner / its admins are submitting position values that still include the already claimed fees.