PUBLIC

# Code Assessment

of the Enzyme Protocol v4 Sulu Smart Contracts

October 12, 2021

Produced for

enzyme

by

CHAINSECURITY

# Contents

# 1 Executive Summary

Dear Enzyme Team,

First and foremost we would like to thank Avantgarde Finance for giving us the opportunity to assess the current state of their Enzyme Protocol v4 Sulu system. This document outlines the findings, limitations, and methodology of our assessment.

The review up to the intermediate report was done in two phases. After the intermediate report all raised issues have been addressed.

Overall, the implementation and its documentation are of a high standard. Apart from the new functionality, the codebase is largely unchanged from the previous release except for some refactoring.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

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 | 2 |
| • Code Corrected | 1 |
| • Acknowledged | 1 |
| Low-Severity Findings | 6 |
| • Code Corrected | 3 |
| • Code Partially Corrected | 1 |
| • Acknowledged | 2 |

# 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 Protocol v4 Sulu repository based on the documentation files. The main audit resulting in the main intermediate was performed in two parts with a gap in between. The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 13 August 2021 | f28abc54f90940d7c2a82f6042b2d7bf857659c1 | Initial Version |
| 2 | 28 September 2021 | 163378feeb694bfcfb1b8db74628f6c3e700d9a0 | Start of 2nd phase |
| 3 | 12 October 2021 | e99215b3515f8266ca0b3228a7fbb7ed827c937d | After Intermediate Report |

Although the audit covers the full Sulu release of Enzyme, many parts of the code have already been covered in the original audit of Enyzme V2. Open Issues / Notes reported in the original audit of Enzyme V2 still apply and are not repeated in this report. For the solidity smart contracts, the compiler version `0.6.12` was chosen. This outdated compiler version has been chosen explicitly due to resue of contracts / code parts of Enzyme V2.

For this report, the main focus has been on:

- External Positions
- Fund Reconfiguration
- Protocol Fees
- Specific Asset Redemption
- Gas relaying
- New fees, new policies
- Reviewing how the new features work with the already audited part

### 2.1.1 *Excluded from scope*

Open Issues / Notes reported in the original audit of Enzyme V2 still apply and are not repeated in this report.

## 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.

Sulu is the next iteration of the Enzyme system. For a full system overview please refer to the original audit report of Enzyme v2. The additional parts of the new version are:

### 2.2.1 Protocol Fees

Sulu introduces protocol fees for the funds. The fees correspond to a percentage of the assets under management. Fees are first accrued in form of shares of the fund before these shares are burned together with burning an appropriate amount of `MLN` tokens.

The fee-mechanism is described as follows:

- After a fund receives a deposit, has shares redeemed or migrates to a new release or configuration, an amount of fee shares is minted for the protocol.
- The fund manager can burn these fee shares only after burning an amount of `MLN` tokens. Since there is no mechanism to enforce that the fund owns `MLN` tokens to burn, an inflated number of fee shares is minted. This aims to incentivise the managers to burn the `MLN` tokens and thereby burning these shares.

A fund can enable automatic fee payback by setting `autoProtocolFeeSharesBuyback` to true. Note that a failure in fee payback should be caught and not cause a transaction to revert.

### 2.2.2 External Positions

The Sulu release includes a new External Position Manager that allows funds to hold external positions.

External positions are managed through the `ExternalPositionManager` which is an extension similar to the `IntegrationManager`. All interactions with external positions of a fund work through the comptroller's call on extension functionality. For a vault to hold an external position, first a new ExternalPosition contract needs to be deployed. This is handled by the `ExternalPositionProxyFactory` contract.

The actions for an external position are:

- Creation/Deployment of a new external position
- Call execution on an external position. This call takes place through an action from the vault itself.
- Removal of an external position
- Reactivation of an existing external position

Each external position is characterised by a type. Each type corresponds to a specific parser which is responsible for appropriately crafting the arguments required for the actions to execute. Moreover, a specified library implements the logic that allows the interaction between the external position and the external protocol e.g., Compound.

Note that the functionality allowing the removal and later reactivation of an external position needs to be considered carefully. There is a concerns regarding an opportunistic fund manager to hide value of the fund in a temporarily deactivated external position. The result will be that the asset values of the fund are underestimated which translates to an underestimated share price.

One type of external position is currently supported, Compound Debt positions.

When a fund borrows assets from a lending platform such as `Compound` one needs to put up collateral. As such assets are no longer freely transferable, these positions must be handled independently from the rest of the vault's assets, hence the external position is used. The following actions are supported with this external position:

All of the following actions can only be done by a privileged account able to pass the `canManageAssets()` function of the fund.

- `addCollateralAssets`

This functions signals to Compound that the specified ctoken assets held by this external position are intended to be used as collateral by calling the Compounds comptrollers `enterMarket()` function. During this action, depending on the parameters, collateral (compound ctokens), can be transferred from the vault to the external position.

- `removeCollateralAssets`

Can be used to return free collateral from the external position to the vault.

- `borrowAssets`

Allows to borrow assets of compound given there is enough collateral at the external position. The borrowed assets are transferred to the vault.

- `repayBorrowedAsets`

During this action, depending on the parameter assets are transferred from the vault to the external position and then onwards to Compound to repay the debt.

- `claimComp`

Using the Compound system makes an account eligible for a reward in the form of COMP tokens. The fund can claim the compensation using this action. The COMP tokens are transferred to the vault.

The compound debt position lib implementation keeps track of the borrowed and collateral assets.

Note that Compound debt positions have to be overcollateralized. Should the collateralization of the borrowed assets drop below a certain ratio, the position may be liquidated. This means the borrowed position is repayed by a third party agent collecting part of the collateral. The amount depends on the current market price plus a liquidation incentive.

### 2.2.3 Reconfiguration of a Vault Within a Release

Reconfiguration in the Enzyme protocol constitutes the deployment of a new Comptroller and the change in the fees and policies configuration. In the previous iteration of the Enzyme protocol, extensive reconfiguration of a fund was only available between releases. The reconfiguration was taking place during a migration of a fund from an old to the newest release. Sulu allows managers to reconfigure their funds within releases.

Similar to migrations, reconfigurations start with the creation of a request on the `FundDeployer` of the fund. A new `ComptrollerProxy` is then deployed. After some time specified by the request has passed, an authorized entity is allowed to call `executeReconfiguration` which destroys the previous comptroller and sets the newly configured one. Like migrations, a pending reconfiguration can also be canceled by an authorized entity by calling `cancelReconfiguration` which deletes the pending request. Note that the reconfiguration does not involve the dispatcher contrary to the Migration. Moreover, since the `FundDeployer` remains the same within a reconfiguration no hooks to the current `FundDeployer` are implemented.

### 2.2.4 Gas Station Network

The Sulu release introduces support for the gas station network enabling interaction with Enzyme without the need to use Ether to pay for transaction fees. Instead of broadcasting a conventional Ethereum transaction to the Ethereum network, the user signs and sends a meta tx to a relay server of the gas station network. A participant of the gas station network then executes this transaction on the Ethereum network by sending it to a relay hub contract which invokes a paymaster contract handling the compensation for the executor. A trusted forwarder contract checks the users signature before the call to the target is executed.

Two different use cases within Enzyme can be distinguished:

Fund Management:

Transactions in the Ethereum network incur a transaction fee hence administrating a fund in Enzyme may be quite costly. In this release of Enzyme a fund can be configured to allow these transaction to be executed via the gas station network and the transaction fee to be paid using weth of the fund. This may be activated by the funds administrator at any time. Upon activation an individual Paymaster Proxy contract for this fund will be created, facilitating pulling of WETH of the funds holdings to pay for the transaction fees.

Following functions can be called by the administrator or an allowed account having the fund paying the transaction fee:

- All functions of the `Vault`
- `callOnExtension`, `vaultCallOnContract`, `buyBackProtocolFeeShares`, `depositToGasRelayPaymaster` and `setAutoProtocolFeeSharesBuyback` of the comptroller.
- `updatePolicySettingsForFund`, `enablePolicyForFund` and `disablePolicyForFund` of the PolicyManager
- `createReconfigurationRequest`, `executeReconfiguration` and `cancelReconfiguration` of the FundDeployer

The fund manager can send their transaction to execute any of these functions via the gas station network. After each call via the gas station network, this Ether amount may be topped up if selected to do so by the given parameters. Top up can also be initiated manually. During top up an amount of WETH is removed from the fund, unwrapped into Ether and sent to the paymaster in order to make its balance 0.2 ETH. This amount is assumed to be small compared to the total value held by the fund and, thus, the fund is not significantly devalued after each call through GSN. Note that additionally anyone using their own Ether can increase this available balance by directly depositing Ether for this address at the GSN relay hub contract.

Furthermore, investors may also use the services of the gas station network notably to buy or redeem shares. Note that this does not use the Paymaster of the Fund to pay for the transaction fee, rather the user has to use another paymaster contract to pay for the transaction.

### 2.2.5 Policy Adapters

Sulu significantly extends the implemented policies from its previous version. In the current system the following hooks are implemented. `PostBuyShares`, `PostCallOnIntegration`, `PreTransferShares`, `RedeemSharesForSpecificAssets`, `AddTrackedAssets`, `RemoveTrackedAssets`, `CreateExternalPosition`, `PostCallOnExternalPosition`, `RemoveExternalPosition`, `ReactivateExternalPosition`.

The policies implemented by the system are:

- `AllowedAdapterIncomingAssetsPolicy`: limits the allowed incoming assets after a call to an integration.
- `AllowedAdaptersPolicy`: limits the possible adapters that can be used by the system. Note, that by default all the adapters are accessible by all vaults if this policy is not enabled.
- `AllowedExternalPositionTypesPolicy`: limits the possible external-position types that can be created and reactivated by a fund.
- `CumulativeSlippageTolerancePolicy`: limits the percentage the value of a fund can change in specific time window after interacting with an external protocol through an adapter.
- `GuaranteedRedemptionPolicy`: enforces a time window where no interaction with redemption blocking adapters such as the synthetix adapter can take place in order to guarantee redemption in that window.
- `OnlyRemoveDustExternalPositionPolicy`: allows removing of external positions only in case the fund holds negligible amount. It is invoked after the removal of an external position. This policy will try evaluate the value of the external position. In case it fails to get the rate of an asset it allows

a timelock to be initiated. After a specified time window has passed, the evaluation will skip the problematic asset, if there is still no rate available for it.

- `OnlyUnrtackDustOrPricelessAssetsPolicy`: allows the removal of assets the balance of which is considered negligible. The implementation also follows the same logic as the `OnlyRemoveDustExternalPosition` giving users the ability to skip the evaluation of a problematic asset.
- `AllowedAssetsForRedemptionPolicy`: limits the assets that can be redeemed by `redeemSharesForSpecificAssets`.
- `MinAssetBalancesPostRedemptionPolicy`: enforces a minimum amount that should be left in the fund after specific asset redemption.
- `AllowedDepositRecipientsPolicy`: limits the accounts that can buy shares to a whitelist of accounts.
- `AllowedSharesTransferRecipientsPolicy`: limits the recipients of share transferring to the ones defined by a list set by the owner of the fund.
- `MinMaxInvestmentPolicy`: Limits the amount of denomination asset amount that can be sent to a fund in order to mint shares.

Note that all the policies that are based on whitelisting or blacklisting addresses make use of the `AddressListRegistry` contract. This contract allows sharing lists of addresses among different funds and adapters. This means that a fund that makes use of an adapter can use a list compiled of another user to handle whitelisting. This requires from the fund managers to trust that the managers of lists they use.

### *2.2.6 Transferable Shares*

Starting from this release the share tokens become transferrable. Note that all transfers must respect the sharesActionTimelock, transfers are not possible should an account be within the timelock. Policies triggering on related hooks may prevent or limit transfers. To ensure shares of a fund are freely transferrable (apart from the sharesActionTimelock), a fuse `sharesAreFreelyTransferable` can be set which prevents policies on transfers.

### *2.2.7 Chainlink Price Oracle*

Sulu uses the latest `AggregatorV3` from Chainlink. This means that the logic of the Chainlink Price Feed has been modified to comply with the Chainlink's API i.e., `AggregatorV3Interface`.

## 2.3 Trust Model & Roles

The Trust Model undergoes a major change with the Sulu Release. Adapters of the Integration Manager, Fees and Policies no longer have to be whitelisted by the protocol to be used in funds. Fund Manager can now use arbitrary adapters, policies and fees. The new trust model relies on the extensive use of policies safeguarding the actions of the fund. Investors need carefully check the policies of a fund prior to investing.

Moreover, the integration with GSN allows owners to execute authorized actions for the vault using vault's ETH instead of paying by themselves. This means that fund investors should trust fund's owner to not intentionally execute repeatedly actions that would drain the ETH held by the fund.

# 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 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 associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. 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 the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than 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 severities. 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 any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- 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 | 1 |

- Funds Can Avoid Paying Protocol Fees Acknowledged

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

- Untracked WETH Code Partially Corrected
- AddressListRegistry Gas Inefficiency Acknowledged
- Redundant Check Acknowledged

## 5.1 Funds Can Avoid Paying Protocol Fees

Design Medium Version 1 Acknowledged

Consider a fund that has not enabled `autoProtocolFeeSharesBuyback` and assume that over time the protocol fee reserve has accrued a sizeable amount of shares of this fund.

The fund, through its manager, can avoid paying protocol fees by employing the following vector:

1. Move all assets of the vault to an external position. Currently, with only the CompoundDebtPosition available as an external position, this means exchanging all of the vaults holdings into a cToken and transferring it to the external position.
2. Remove the external position. Note that, at this point, the GAV of the fund is close to 0.
3. Call `buyBackProtocolFeeShares` which calculates a really low price per share. This means that only a small amount of `MLN` tokens needs to be burnt to burn the protocol fee shares and, thus, pay back the fee.
4. Reactivate the external position and move the funds back.

Note that a similar vector can be used by the fund manager to avoid minting protocol fees while migrating or reconfiguration the fund.

The opportunistic behavior of the manager against the users of the fund is well documented. However, adversarial actions against the protocol are not mentioned.

Note that the underlying problem, a manipulated lower GAV due to hiding assets of the fund in a removed external position can also be abused by the fund manager to buy shares for a low price.

---

**Acknowledged:**

Avantgarde Finance responded:

> As the audit team importantly noted, this issue only potentially affects the protocol fee amount ultimately burned, and does not impact end users of the protocol. Hence, rather than changing the core contracts for this release to protect against the reported deviant behavior, we have decided to combine the monitoring of blatant protocol fee violations with potential on- and off-chain penalties.
>
> Deviant behavior can monitored off-chain by comparing each shares buyback event with the last known share price.
>
> If the Council assesses that there is a blatant attempt to evade protocol fees, it would be possible, for example, to restrict buying back shares by upgrading the ProtocolFeeReserveProxy contract to disallow particular funds.
>
> We can reevaluate for subsequent releases whether or not to prevent this behavior at the core protocol level.

## 5.2 Untracked WETH

**Correctness** **Low** **Version 2** **Code Partially Corrected**

A fund owner can withdraw the Ether that has been deposited to the paymaster account by calling `GasRelayPaymasterLib.withdrawBalance`. At this point Ether will be transferred to the vault and wrapped into `WETH`. However, there is no guarantee that `WETH` is a tracked asset for the fund.

---

**Code Partially Corrected:**

When `ComptrollerLib.shutdownGasRelayPaymaster` is called, `WETH` is added to the tracked assets.

Avantgarde Finance responded:

> It is highly unlikely that a fund using a paymaster would not have WETH as a tracked asset. Still, we have added a call to track WETH as a tracked asset when shutdownGasRelayPaymaster() is called from the ComptrollerProxy. It is more difficult - and best not - to attempt to validate whether WETH is a tracked asset when calling withdrawBalance() directly from the paymaster lib, since it can be called after a fund has migrated to a new release, at which point, the interface of its new VaultLib should not be assumed.

## 5.3 `AddressListRegistry` Gas Inefficiency

**Design** **Low** **Version 2** **Acknowledged**

`AddressListRegistry` stores `ListInfo` using the mapping `itemToIsInList`. The type of the mapping is `mapping(address => bool)`. However, it would be more gas efficient to set its type to `address => uint256` which omits the masking operation required to handle the boolean values.

---

**Acknowledged:**

Avantgarde Finance responded:

> We acknowledge the technical efficiency, but in practice the gas savings is insignificant within the context of the protocol, and the use of `bool` is more intuitive.

## 5.4 Redundant Check

Design | Low | Version 1 | Acknowledged

In `FundDeployer.__redeemSharesSetup`, the following snippet exists:

```
    } else if (postFeesRedeemerSharesBalance < preFeesRedeemerSharesBalance) {
         ...
         preFeesRedeemerSharesBalance.sub(postFeesRedeemerSharesBalance)
     );
}
```

Note that the use of sub is redundant in this case since it holds: `postFeesRedeemerSharesBalance < preFeesRedeemerSharesBalance`.

---

**Acknowledged:**

Avantgarde Finance responded:

> Currently, we generally use SafeMath for math operations rather than making judgements about where or where not to use it.

# 6 Resolved Findings

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

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 | 1 |

- redeemSharesForSpecificAssets Fails For Derivatives Code Corrected

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

- Incorrect redemptionWindowBuffer Check Code Corrected
- Shadowed Constant Code Corrected
- Missing Indexes in Events Code Corrected

## 6.1 `redeemSharesForSpecificAssets` Fails For Derivatives

Correctness Medium Version 1 Code Corrected

In Sulu, users are allowed to redeem specific assets. According to the documentation:

> The redeemer specifies one or multiple of the VaultProxy's ERC20 holdings along with the relative values of each to receive (for a total of 100%).

However, if an ERC20 token which represents a derivative is specified as a payout asset then the redemption will fail. The call in `__payoutSpecifiedAssetPercentages` fails for non primitive assets due to the require statement shown below in the `calcCanonicalAssetValue` of the `valueInterpreter`:

```
payoutAmounts_[i] = IValueInterpreter(getValueInterpreter()).calcCanonicalAssetValue(
    denominationAssetCopy,
    _owedGav.mul(_payoutAssetPercentages[i]).div(ONE_HUNDRED_PERCENT),
    _payoutAssets[i]
);

function calcCanonicalAssetValue(
    ...

    require(
        isSupportedPrimitiveAsset(_quoteAsset),
        "calcCanonicalAssetValue: Unsupported _quoteAsset"
    );

    ...
```

Note that simply removing the requirement is not enough since the `ValueInterpreter` can only handle conversions to primitive assets due to the implicit requirement that the quote asset has a Chainlink price feed.

**Code Corrected:**

`ValueInterpreter.calcCanonicalAssetValue` now supports the conversion from a primative asset to a derivative asset. This is done by calculating the price of the derivative asset against the primative one and the inverting.

## 6.2 Incorrect redemptionWindowBuffer Check

Correctness Low Version 2 Code Corrected

Certain actions through adapters e.g. exchanging Synths through the Synthetix adapter may block the transfer of the asset for a period of time. The GuaranteedRedemptionPolicy ensures that a redemption blocking adapter is not used during the redemption window nor a buffer period before the start of the guaranteed redemption window. This ensures redemption is possible during a guaranteed time window every day.

```
uint256 latestRedemptionWindowStart = calcLatestRedemptionWindowStart(
        redemptionWindow.startTimestamp
    );


// A fund can't trade during its redemption window, nor in the buffer beforehand.
// The lower bound is only relevant when the startTimestamp is in the future,
// so we check it last.
if (
    block.timestamp >= latestRedemptionWindowStart.add(redemptionWindow.duration) ||
    block.timestamp <= latestRedemptionWindowStart.sub(redemptionWindowBuffer)
) {
    return true;
}
return false;
```

The comment describing the code is not entirely accurate. Three cases have to be distinguished

- I. A fund can't trade during its redemption window
- II.a. A fund can't trade in the buffer before the next redemption window
- II.b. If startTimestamp is in the future, a fund can't trade in the buffer before the first redemption window

Note that `calcLatestRedemptionWindowStart()` returns either the start timestamp of the latest redemption window or, in case startTimestamp is still in the future, the startTimestamp.

The current code checks condition (I) and (IIb) but does not check (IIa). Hence, in case we are past `startTimestamp` and there exists a `latestRedemptionWindowStart` timestamp in the past, a trade in the buffer window before the start of the next guaranteed redemption period is not prevented and such a trade may prevent redemption in the guaranteed redemption timeframe.

**Code Corrected:**

In the current implementation, the `startTimestamp` is required to be in the past. Moreover, the `redemptionWindowBuffer` is now subtracted from the `latestRedemptionWindowStart` with the addition of one day.

```
if (
    block.timestamp > latestRedemptionWindowStart.add(redemptionWindow.duration) &&
    block.timestamp < latestRedemptionWindowStart.add(ONE_DAY).sub(redemptionWindowBuffer)
) {
    return true;
}
```

## 6.3 Shadowed Constant

Design Low Version 2 Code Corrected

UniswapV2PoolPriceFeed inherits UniswapV2PoolTokenValueCalculator. Both contracts define a constant `uint256 private constant POOL_TOKEN_UNIT = 10**18;`.

---

**Code Corrected:**

The shadowing variable has been removed.

## 6.4 Missing Indexes in Events

Design Low Version 1 Code Corrected

The `PreRedeemSharesHookFailed` event in `ComptrollerLib` could use indexes for `address redeemer`. Moreover, the `bytes FailureReturnData` of `PreRedeemSharesHookFailed` and `BuyBackMaxProtocolFeeSharesFailed` could be indexed since it could facilitate queries for specific errors.

---

**Code Corrected:**

The missing indexes have been added.

# 7 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.

## 7.1 Overestimation Of Fund's Value Under Pending Liquidation

**Note** **Version 1**

During the calculation of the GAV of a fund, the value of the collateral held by an external position is taken into consideration. The calculation takes into account the fact that borrowed amount of an external position is to be returned and, thus, this amount is subtracted from the total collateral held. However, the calculation ignores a potential liquidation.

Assume an external position that holds 100 cDAI and has borrowed 75 dollars worth of ETH with a collateral factor of 75%. Assume now that the value of ETH has increased so that the external owes 80 dollars. When the GAV is calculated the external position will be evaluated as 100 - 80 = 20 dollars. Since the position is undercollateralized, a liquidation could be triggered. Note that during liquidations, users are incentivized to pay back the borrowed amount with an 8% discount for the collateral. When the liquidation takes place then the real value of the external position will be roughly 100 - 86.4 = 13.6. Users should be aware of that behavior which might lead to fluctuations in the GAV of the fund. Moreover, the front end of Enzyme should indicate potentially undercollateralized external positions to users prior to investing.

## 7.2 Reverting Relayed Call Paid By The Fund

**Note** **Version 1**

According to the GSN protocol, in case `GasRelayPaymaster.preRelayedCall` fails then the execution is aborted. However, there might be the case where the `preRelayedCall` succeeds but the relayed transaction fails. In this case, the fund will still pay for the gas. An interesting case is the following. The `preRelayCall` requires the original `_relayRequest.request.from` to be an authorised entity for the vault i.e., the owner or an asset manager or a migrator. However, some of the authorised calls allowed by the `preRelayedCall` further restrict the allowed entities. For example, a call to an integration is limited to only the owner and the asset manager. This means that in case the migrator tries to execute this function, the transaction will fail but paymaster will pay for the gas. We assume, however, that the migrator is a trusted role who will not act against the system.

## 7.3 Sandwiching Authorized Actions

**Note** **Version 1**

An authorized user for a fund can use GSN to execute authorized actions. Is important to note that due to the fact that the relayer acts as an intermediary, it is easier for them to sandwich these transactions. For example, a relayer can sandwich the `buyBackProtocolFeeShares` and make a profit. Notice that when buying back shares the value of the shares increases since the shares which correspond to the protocol fee are burnt.