PUBLIC

Code Assessment

of the Sulu Extensions XV Smart Contracts

July 16, 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 Sulu Extensions XV according to Scope to support you in forming an opinion on their security risks.

Enzyme Foundation implements a price feed for EETH and a wrapper for Chainlink-like oracles with non-standard decimals. Additionally, the GSN integration is adapted. A new policy restricting redeem-for-specific-assets is introduced along with a peripheral FIFO contract that can redeem-for-specific-assets. Further, an asset manager contract is introduced that limits the share price loss an asset manager can cause. For integrations, a Pendle Finance external position and a Swell adapter are introduced.

The most critical subjects covered in our audit are asset solvency, functional correctness and integration with external systems and Enzyme's core system. Security regarding all the aforementioned subjects is high.

The general subjects covered are error handling, specification and trustworthiness. Security regarding all the aforementioned subjects is high. Note that the trust model for using GSN slightly changed, see the note Gas relay paymaster will fund arbitrary calls.

In summary, we find that the codebase provides a good 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	1
• Code Corrected	1
Medium-Severity Findings	3
• Code Corrected	3
Low-Severity Findings	3
• Risk Accepted	3

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 Sulu Extensions XV repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

dev branch - New Modules

V	Date	Commit Hash	Note
1	15 March 2024	b5ea6ad542e2122fdb77a6fb18c8e0da49285193	Initial Version
2	19 March 2024	7c97f9cdfa65df256f832f70bd10702ef1902b5b	GSN, Queue, Throttled Manager
3	22 March 2024	cbb018762117e4995b447d7f7655be26cf424a04	Wrappers and Pendle March
4	26 March 2024	fe18ecc2a7dd6d40a7da037e13f457de63ec1ca0	Partial Fixes and Int256 Lib
5	22 April 2024	473d4b2e8771042e67ab0a6b7bdfb26dcfcddbb	After Intermediate Report
6	29 April 2024	18b840207a42c86eeeb20a7b1b923f7f61a13b0a	Final Changes
7	15 July 2024	1e3d9911d0e94c688198fc19fb00ccbf8c40f89b	Bug Fix

v4 branch - Paymaster Changes

V	Date	Commit Hash	Note
1	22 March 2024	ba3060214226e42892a541db77ef7d4cc9ab54e4	Changes to Paymaster

For the solidity smart contracts, the compiler versions `0.8.19` and `0.6.12` (only for gas relay paymaster scope) were chosen.

The EtherFi Price Feed scope is:

```
contracts/external-interfaces/IEtherFiWrappedEth.sol
contracts/release/infrastructure/price-feeds/derivatives/feeds/EtherFiEthPriceFeed.sol
```

The scope for the Chainlink-like price feed with non-standard decimals is:

```
contracts/external-interfaces/IChainlinkAggregator.sol
contracts/release/infrastructure/price-feeds/primitives/NonStandardPrecisionSimulatedAggregator.sol
```

The scope for the single redeem-for-specific-assets caller policy is:

```
contracts/release/extensions/policy-manager/policies/utils/0.8.19/AddressListRegistryPolicyBase.sol
contracts/release/extensions/policy-manager/policies/current-shareholders/AllowedRedeemersForSpecificAssetsPolicy.sol
```

The scope for the new GSN Recipient mixin is:

```
contracts/utils/0.8.19/gas-station-network/GSNRecipientMixin.sol
```

The scope for the single asset redemption queue is:

```
contracts/utils/0.8.19/NonUpgradableProxy.sol
contracts/persistent/single-asset-redemption-queue/ISingleAssetRedemptionQueue.sol
contracts/persistent/single-asset-redemption-queue/SingleAssetRedemptionQueueFactory.sol
contracts/persistent/single-asset-redemption-queue/SingleAssetRedemptionQueueLib.sol
```

The scope for the share price-throttled asset manager is (including multicall asset manager template):

```
contracts/persistent/smart-accounts/share-price-throttled-asset-manager/ISharePriceThrottledAssetManagerLib.sol
contracts/persistent/smart-accounts/share-price-throttled-asset-manager/SharePriceThrottledAssetManagerFactory.sol
contracts/persistent/smart-accounts/share-price-throttled-asset-manager/SharePriceThrottledAssetManagerLib.sol
contracts/persistent/smart-accounts/utils/MultiCallAccountMixin.sol
contracts/persistent/smart-accounts/utils/interfaces/IMultiCallAccountMixin.sol
```

The scope for the Pendle Finance external position is:

```
contracts/external-interfaces/IPendleV2Market.sol
contracts/external-interfaces/IPendleV2MarketFactory.sol
contracts/external-interfaces/IPendleV2PrincipalToken.sol
contracts/external-interfaces/IPendleV2PtOracle.sol
contracts/external-interfaces/IPendleV2Router.sol
contracts/external-interfaces/IPendleV2StandardizedYield.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/IPendleV2Position.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/PendleV2PositionDataDecoder.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/PendleV2PositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/PendleV2PositionParser.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/bases/PendleV2PositionLibBase1.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/markets-registry/IPendleV2MarketRegistry.sol
contracts/release/extensions/external-position-manager/external-positions/pendle-v2/markets-registry/PendleV2MarketRegistry.sol
```

The following contracts diff to Pendle Finance's contracts are in scope (but not the logic itself):

```
contracts/utils/0.8.19/pendle/adapted-libs/Errors.sol
contracts/utils/0.8.19/pendle/adapted-libs/LogExpMath.sol
contracts/utils/0.8.19/pendle/adapted-libs/MarketMathCore.sol
contracts/utils/0.8.19/pendle/adapted-libs/MiniHelpers.sol
contracts/utils/0.8.19/pendle/adapted-libs/PMath.sol
contracts/utils/0.8.19/pendle/adapted-libs/PYIndex.sol
contracts/utils/0.8.19/pendle/adapted-libs/PendleLpOracleLib.sol
contracts/utils/0.8.19/pendle/adapted-libs/PendlePtOracleLib.sol
contracts/utils/0.8.19/pendle/adapted-libs/SYUtils.sol
contracts/utils/0.8.19/pendle/adapted-libs/interfaces/IPendleV2Market.sol
contracts/utils/0.8.19/pendle/adapted-libs/interfaces/IPendleV2StandardizedYield.sol
contracts/utils/0.8.19/pendle/adapted-libs/interfaces/IPendleV2YieldToken.sol
```

The scope for the Swell adapter (including the template for wrap/unwrap adapters) is:

```
contracts/external-interfaces/ISwellSweth.sol
contracts/release/extensions/integration-manager/integrations/adapters/SwellStakingAdapter.sol
contracts/release/extensions/integration-manager/integrations/utils/0.8.19/bases/GenericWrappingAdapterBase.sol
contracts/release/extensions/integration-manager/integrations/utils/IntegrationSelectors.sol
```

The scope for the library for 256-bit integer arrays is:

```
contracts/utils/0.8.19/Int256ArrayLib.sol
```

The scope for the GSN Relay Paymaster is:

```
contracts/release/infrastructure/gas-relayer/GasRelayPaymasterLib.sol
contracts/release/infrastructure/gas-relayer/IGasRelayPaymaster.sol
```

2.1.1 Excluded from scope

Any contracts inside the repository that are not mentioned in `Scope` are not part of this assessment. All external libraries and imports are assumed to behave correctly according to their high-level specification, without unexpected side effects.

More derived usage of template contracts and mixins is not in scope, with the exception of the more derived contracts in scope.

The selection of deployment parameters is not in scope (e.g. valid duration ranges for Pendle's TWAP).

The correctness of external systems is not in scope.

The libraries copied from Pendle are out-of-scope. However, the formatting changes applied for compatibility with the codebase are in scope.

The correct usage of the Pendle market registry is out of scope. Fund owners should be well-informed about the proper usage of the registry.

Tests and deployment scripts are excluded from the scope.

2.2 System Overview

This system overview describes the most recently received version 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 derivate and a primitive price feed for EtherFi's eETH staked ETH token and Chainlink-like price feeds with non-standard decimals (e.g. RedStone Classic oracles, initially intended for pricing EtherFi's weETH token) respectively. Further, a new policy is implemented that restricts the redeem-for-specific-assets callers. Leveraging the new policy, a peripheral contract is implemented that offers a FIFO queue for redeem-for-specific-assets callers. Further, a peripheral contract, acting as manager and building on a new multicall template, is introduced to allow the rate-limiting of losses created by manager actions. Both peripheral contracts extend a newly introduced GSN mixin. Note that the gas paymaster has been adapted to support refunding of additional callers and calling arbitrary contracts so that the actions are performed through the asset manager contracts (e.g. the share price throttled asset manager contract). Last, an external position for integrating with Pendle Finance to provide liquidity or get fixed yield, and a simple wrap/unwrap template along with Swell- and Diva-specific implementations are provided.

2.2.1 Price Feed: EtherFi eETH

The EtherFi eETH price feed prices eETH tokens in weETH tokens with the derivative price feed function `calcUnderlyingValues()` by converting the eETH amount with `weETH.getWeETHByeETH()` that effectively computes the amount with the liquidity pool's `sharesForAmount()` function. The only supported asset defined by `isSupportedAsset()` is eETH.

2.2.2 Price Feed: Chainlink-like with non-standard decimals

Some Chainlink-like price sources deviate from the decimal standards used by Chainlink. This oracle scales the price of the underlying feed to 18 decimals for ETH oracles and to 8 decimals for USD oracles. Initially, this is intended to be used with RedStone Classic oracles.

2.2.3 Policy: Allowed redeem-for-specific-assets caller

This policy restricts calls to `ComptrollerLib.redeemSharesForSpecificAssets()` to callers that are present in at least one of the policy configuration specified lists in the address list registry.

2.2.4 Mixin: GSN Recipient

Simple mixin for the GSN integration which is similar to the previously audited one. However, multiple trusted forwarders are allowed but need to be specified in a list in the address list registry.

2.2.5 Periphery: Single-Asset Redemption Queue

`SingleAssetRedemptionQueueLib` implements a FIFO queue for calls to `ComptrollerLib.redeemSharesForSpecificAssets()` for a given asset. Note that it is expected that it is the only address allowed to redeem-for-specific-assets (could be enforced with the new policy added). Users can then request a single-asset redemption through `requestRedeem()` which puts them in the queue. Managers of the contract or the owner of the fund can initiate a redemption for a batch in the queue with `redeemFromQueue()` which calls `ComptrollerLib.redeemSharesForSpecificAssets()` and distributes the received tokens to the redeemers. note that typically, the FIFO (First In, First Out) principle is followed, except for privileged addresses. These addresses may include redemptions to bypass the FIFO order, provided the redemption amount does not exceed a certain threshold. Further, users can withdraw their unsuccessful redemption requests with `withdrawRequest()`. Note that the queue can be `shutdown()` by the fund owner which disables redemption requests and redemption executions (users can withdraw their requests). Further, the fund owner can add and remove managers with `addManagers()` and `removeManagers()` respectively. Additionally, the fund owner can set the threshold with `setBypassableSharesThreshold()` and can change the asset to withdraw with `setRedemptionAsset()`.

The `SingleAssetRedemptionQueueFactory` allows deploying `NonUpgradeableProxy` contracts that point to the lib.

2.2.6 Periphery: Share price-throttled asset manager

`SharePriceThrottledAssetManagerLib` is a `MultiCallAccountMixin` which allows the contract owner to batch-execute calls with `executeCalls()` as is intended to be used assigned an asset manager role. Additionally, it tracks cumulative losses of the fund over a rolling duration `lossTolerancePeriodDuration` and prevents calls if they exceed the `lossTolerance`. Effectively, the `SharePriceThrottledAssetManagerLib``rate-limits the losses coming from asset manager actions. Furthermore, ``shutdown()` can be invoked by the `shutdowner` which sets the `owner` to 0, preventing anyone from calling `executeCalls()`.

The `SharePriceThrottledAssetManagerFactory` allows deploying `NonUpgradeableProxy` contracts that point to the lib.

2.2.7 External Position: Pendle Finance

Pendle Finance is a yield-tokenization platform where users split their yield-bearing tokens into a principal and a yield token - both with given maturities. On top of that, an AMM is built where users can trade their principal tokens and the yield-bearing tokens.

The external position implements the following actions:

1. `BuyPrincipalToken`: Mints the standardized yield token (wrapper around yield-bearing tokens) and trades it on the market to get principal tokens.
2. `SellPrincipalToken`: In case the Pendle contracts are not expired, trades the principal token for standardized yield tokens and burns the standardized yield token. Otherwise, it redeems the principal token for standardized yield tokens and then proceeds to burn.

3. `AddLiquidity`: Mints the standardized yield token and adds one-sided liquidity to the market (in Pendle's router this is implemented as trading and then adding liquidity in a balanced manner).
4. `RemoveLiquidity`: Burns the LP for single-sided liquidity (in Pendle's router this is implemented as burning the LP and then swapping the received principal tokens to standardized yield tokens) and burns the received standardized yield tokens.
5. `ClaimRewards`: Claims the rewards from a list of markets and transfers them to the vault.

Since the position does not take on any debt, `getDebtAssets()` returns empty arrays. `getManagedAssets()` evaluates the tracked LP and principal tokens with their underlying asset by leveraging Pendle Finance's library functions that make use of the markets' TWAP oracle functionality.

Note that the validation of markets is performed against the `PendleV2MarketRegistry` which is a new registry. The registry supports the `updateMarketsForCaller()` which allows the vault to register Pendle markets. It works as described below:

1. A market can be added with a duration (used for TWAPs). The duration can be updated for a market.
2. The market's PT automatically points to the most recent added/updated market (see 1.). That assumes that a market always points to the same market.
3. A market can be removed by specifying the duration to be zero.
4. A PT is automatically removed if the pointed-to market is removed.

2.2.8 Integration: Simple wrapping and unwrapping actions

A generic and abstract adapter for `wrap()` and `unwrap()` operations, `GenericWrappingAdapterBase`, is introduced that can convert an underlying asset to a derivative and vice-versa. Hence, `parseAssetsForAction()` supports these operations where the spent assets (transferred to the integration) and minimum incoming assets are defined by the input provided by managers. Further, note that the adapter can use native ETH if the integrated protocol requires so (defined by the immutable `USE_NATIVE_ASSET`). The internal functions `__wrap()` and `__unwrap()` revert by default but should typically be overridden by more derived contracts to define the wrapping operations according to the integrated protocol.

2.2.8.1 Swell ETH

The `SwellStakingAdapter` leverages the `GenericWrappingAdapterBase` to allow for wrapping ETH into swETH, the staked ETH token of Swell. Note that unwrapping remains unavailable due to the staking withdrawal queue but wrapping is implemented by calling swETH's `depositWithReferral()` function.

2.2.9 Periphery: GSN Paymaster Changes

The `GasRelayPaymasterLib` has been adapted. Namely, the following changes have been made:

1. The fund owner can add and remove additional relay users with `addAdditionalRelayUsers()` and `removeAdditionalRelayUsers()`. That is to support callers in asset manager contracts (e.g. in the share price-throttled one, see Periphery: Share price-throttled asset manager).
2. The value of a call cannot be greater than zero.
3. The target and the selectors are not validated anymore. Hence, it is possible to call arbitrary functions on arbitrary contracts while the paymaster is paying.
4. Last, the paymaster integrates with GSN itself. Namely, it is a call GSN recipient itself.

2.2.10 Roles and Trust Model

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

In general, we assume Enzyme only interacts with normal ERC-20 tokens that do not have multiple entry points, callbacks, fees-on-transfer, or other special behaviors.

Fund owners and asset managers are generally fully trusted for a fund. However, their powers can be limited through the fund's settings. The funds' settings/policies are assumed to be set up correctly for the intended configuration/usage. Fund owners are expected to only register suitable Pendle markets and principal tokens (see Resolved Findings for examples of bad markets). Further, they are expected to remove all PTs before unsupporting them. Note that for the single asset redemption queue, funds could be locked.

The managers are expected to regularly claim the fees and to pause the position if under-/overvaluation of the fund occurs.

Governance is fully trusted and expected to not only behave honestly but also to fully understand the systems they are interacting which includes choosing appropriate parameters.

External systems are expected to work correctly and as expected. Further, Pendle Finance expects yield-bearing tokens to only increase in value and never decrease. Hence, funds utilizing the EP are inheriting that assumption.

All external systems are trusted to behave as expected and correctly.

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 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	3

5.1 DoSed Queue With Reverting Transfers

Design Low Version 1 Risk Accepted

CS-SUL15-003

The `SingleAssetRedemptionQueueLib.redeemFromQueue()` could revert in case a token transfer fails and thus block the queue from processing. While reverts typically are not expected, some tokens may revert due to blacklisting (e.g. USDC blacklisted a recipient). Ultimately, processing the queue may be blocked.

However, note that the fund owner may correct such a scenario by temporarily increasing the bypass-threshold to allow bypassing the blocking request.

Risk Accepted:

Enzyme Foundation accepts the risk with the following statement: :

> As noted in the issue, the fund owner can raise the bypassable shares amount to skip such requests as-needed

5.2 Reinitializing Throttled Manager

Correctness Low Version 1 Risk Accepted

CS-SUL15-004

During `init()`, `vaultProxyAddress` is set. To prevent `init` from being called again, there is a check that `vaultProxyAddress` is not 0. However, the value (`_vaultProxyAddress`) to which is set could be `0x0`. Thus, the check can be bypassed.

Risk Accepted:

Enzyme Foundation accepts the risk with the following statement:

> A fund owner must verify the configuration of the deployed proxy before adding it as an asset manager.

5.3 Remainder in Pendle Actions

Design | Low | Version 1 | Risk Accepted

CS-SUL15-005

When SYs are traded (adding liquidity or buying PT), the `ApproxParams` struct needs to be provided to the Pendle router. That is due to the design of Pendle's AMM which cannot trivially trade exact amounts of SY. Hence, an approximation of target PT amounts is done to estimate the exact output PT. As a consequence, SY may remain in the market which could be retrieved with `skim()`.

Risk Accepted:

Enzyme Foundation accepts the risk with the following statement: :

> This is essentially considered to be part of overall slippage, which is validated via the user-input min expected amount.

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	1

- Bad eETH Conversion Code Corrected

Medium-Severity Findings	3

- Claiming LPs and PTs as Rewards Code Corrected
- PTs With Untrusted SY Code Corrected
- Queue Redemption Out-of-Bounds Code Corrected

Low-Severity Findings	0
Informational Findings	1

- Page Not Found in Pendle Docs Code Corrected

6.1 Bad eETH Conversion

Correctness High Version 7 Code Corrected

CS-SUL15-010

Enzyme Foundation reported an issue with the conversion of eETH to weETH. Namely, `getEETHByWeETH` was used which converts from weETH to eETH. However, `getWeETHByeETH` is required to convert from eETH to weETH.

Code corrected:

Enzyme Foundation implemented the correct conversion.

6.2 Claiming LPs and PTs as Rewards

Correctness Medium Version 1 Code Corrected

CS-SUL15-001

The `ClaimRewards` action does not validate the markets. As a consequence, managers may claim LPs and PTs as rewards if an input market has a PT or LP held by the position as a reward.

Consider the following example:

1. The position holds an sDAI-PT.
2. A manager deploys a contract that returns sDAI-PT as a reward token.
3. The manager claims rewards and specifies his contract as a market.
4. The sDAI-PT is transferred to the vault and remains untracked.

5. The fund lost in value and shares could be bought cheaper than they should be.

Ultimately, the shares could be devaluated.

Code Corrected:

`ClaimRewards` now filters tracked LPs and PTs out of the rewards.

6.3 PTs With Untrusted SY

Security | Medium | Version 1 | Code Corrected

CS-SUL15-002

Managers could drain user funds by adding funds to an untrusted standardized yield token that is used by a factory-deployed PT.

While managers are generally trusted, their capabilities are limited. For example, the Pendle markets and principal tokens are validated to be created through trusted Pendle factories. However, the underlying standardized yield tokens are not validated in Pendle when a PT/YT pair is deployed. Hence, a malicious asset manager could add funds to a malicious SY so that funds can be stolen.

Ultimately, standardized yield tokens are not validated which could lead to the stealing of funds.

Code Corrected:

A `PendleV2MarketRegistry` is now used in `PendleV2PositionLib` to only allow PTs to be traded on the market that is added to the registry by the user. This ensures that only trusted markets are used for trading PTs.

6.4 Queue Redemption Out-of-Bounds

Correctness | Medium | Version 1 | Code Corrected

CS-SUL15-009

Calls to `SingleAssetRedemptionQueueLib.redeemFromQueue()` can revert due to using the indexes of the underlying storage array for the memory arrays.

More specifically, the queue is designed to have a pointer to the start index and end index of the underlying array. The function operates on the array indexes and not on the position in the queue. Thus, the iteration over the unprocessed redemption requests iterates over the range of array indexes from some `startId` to some `_endId`. While doing so, data relevant to the processed batch of redemption requests is put into memory arrays of the size of the batch. Hence, data for `startId` should be stored at position 0 of the memory for example. However, the code tries to store the data in the memory array at the same position as the ID was in the underlying array of the queue. That leads to out-of-bounds problems.

To summarize, redeeming from the queue will always fail after the first successful execution.

Code corrected:

The array index is now computed as follows: `index = id - startId`, fixing the out-of-bounds accesses.

6.5 Page Not Found in Pendle Docs

Informational | Version 1 | Code Corrected

CS-SUL15-007

The comments in code reference the Pendle documentation. However, the pages are not found.

Code Corrected:

The comments in the code now reference the correct Pendle documentation.

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 Implementing Aggregator Interface

Informational Version 1 Risk Accepted

CS-SUL15-006

The `NonStandardPrecisionSimulatedAggregator` supports the `IChainlinkAggregator` interface but does not implement it.

Risk Accepted:

Enzyme Foundation accepts the risk described.

7.2 Queue Gives Dust to Next Batch Processed

Informational Version 1 Risk Accepted

CS-SUL15-008

The `SingleAssetRedemptionQueueLib` computes the funds to distribute to users in a way where dust can remain in the contract (no dustless computation). Consider the following scenario:

1. Alice requests the redemption of 999 shares.
2. Bob requests the redemption of 1 share.
3. The redemption queue is processed. 999 tokens are received from the redemption.
4. Alice receives `999 * 999 / 1000 = 998` shares.
5. Bob receives `999 * 1 / 1000 = 0` shares.
6. 2 shares remain in the contract.

The dust can then be claimed by the next processing of redemption requests. Ultimately, the last batch could leave some permanent dust in the contract.

However, note that only small amounts will be left in the contract.

Risk Accepted:

Enzyme Foundation accepts the risk described.

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 Aggregator Interface Not Fully Implemented

Note Version 1

The `NonStandardPrecisionSimulatedAggregator` is a wrapper for a subset of the functionality defined by standard Chainlink aggregators. Integrators of the simulated aggregators should be aware of that.

8.2 Gas Relay Paymaster Will Fund Arbitrary Calls

Note Version 1

Users of Enzyme should be aware that the gas relay paymaster now can fund arbitrary calls. Hence, the fund owner, asset managers and the additional addresses could use the paymaster for other purposes. However, the rate limit on the paymaster is active and limits bad behaviour.

8.3 Referral Address Should Not Be Sender

Note Version 1

For the `SwellStakingAdapter`, the referral address should not be the adapter itself as it will cause any wrapping attempt to fail.