PUBLIC

# Code Assessment

## of the Sulu Extensions VII Smart Contracts

November 8, 2022

Produced for

by

# Contents


1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 9
4 Terminology 10
5 Findings 11
6 Resolved Findings 13
7 Notes 15

# 1 Executive Summary

Dear Mona and Sean,

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

Avantgarde Finance implements integrations for staking Balancer LP tokens natively or on Aura and provides price feeds for the staked tokens and Balancer v2 stable pool LP tokens. Further, batching ParaSwap orders with optional individual failures, staking ETH on Kiln, and periphery shares wrapper contracts for arbitrary deposit tokens were introduced.

The most critical subjects covered in our audit are functional correctness, integration with external systems, and access control. Security regarding functional correctness is improvable due to potentially unexpected behaviour, see Unexpected staking of tokens. Security regarding integration with external systems is improvable due to slashing being unhandled for Kiln, see Unhandled stake slashing on Kiln.

The general subjects covered are gas efficiency, documentation, code complexity and error handling. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a good but improvable level of security.

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

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

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

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

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 3 |
| • Code Corrected | 1 |
| • Risk Accepted | 2 |
| Low-Severity Findings | 3 |
| • Code Corrected | 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 VII 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 | 17 October 2022 | de3c114eb27a88aecc5b16aba4029eb8223e5fbf | Initial Version |
| 2 | 7 November 2022 | cdb25834be0a9e55dfd671863d93a0dada473a0e | Second Version |

For the solidity smart contracts, the compiler version `0.6.12` was chosen.

Balancer v2 Stable Pool Price Feed:

```
contracts/release/infrastructure/price-feeds/derivatives/feeds/BalancerV2StablePoolPriceFeed.sol
contracts/release/interfaces/IBalancerV2StablePool.sol
contracts/release/infrastructure/price-feeds/derivatives/feeds/BalancerV2WeightedPoolPriceFeed.sol
contracts/release/interfaces/IBalancerV2PoolFactory.sol
```

Balancer v2 Staking:

```
contracts/release/extensions/integration-manager/integrations/adapters/AuraBalancerV2LpStakingAdapter.sol
contracts/release/extensions/integration-manager/integrations/adapters/BalancerV2LiquidityAdapter.sol
contracts/release/extensions/integration-manager/integrations/utils/bases/BalancerV2LiquidityAdapterBase.sol
contracts/release/infrastructure/price-feeds/derivatives/feeds/AuraBalancerV2LpStakingWrapperPriceFeed.sol
contracts/release/infrastructure/price-feeds/derivatives/feeds/BalancerV2GaugeTokenPriceFeed.sol
contracts/release/infrastructure/staking-wrappers/aura-balancer-v2-lp/AuraBalancerV2LpStakingWrapperFactory.sol
contracts/release/interfaces/IBalancerV2LiquidityGauge.sol
```

External Position: Kiln:

```
contracts/release/interfaces/IKilnDepositContract.sol
contracts/persistent/external-positions/kiln-staking/KilnStakingPositionLibBase1.sol
contracts/release/extensions/external-position-manager/external-positions/kiln-staking/IKilnStakingPosition.sol
contracts/release/extensions/external-position-manager/external-positions/kiln-staking/KilnStakingPositionDataDecoder.sol
contracts/release/extensions/external-position-manager/external-positions/kiln-staking/KilnStakingPositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/kiln-staking/KilnStakingPositionParser.sol
```

Multi-order batching in ParaSwap v5 adapter:

```
contracts/release/extensions/integration-manager/integrations/adapters/ParaSwapV5Adapter.sol
contracts/release/extensions/integration-manager/integrations/utils/IntegrationSelectors.sol
```

Shares wrapper for arbitrary deposit token:

```
contracts/release/peripheral/shares-wrappers/arbitrary-token-phased/ArbitraryTokenPhasedSharesWrapperFactory.sol
contracts/release/peripheral/shares-wrappers/arbitrary-token-phased/ArbitraryTokenPhasedSharesWrapperLib.sol
contracts/release/peripheral/shares-wrappers/arbitrary-token-phased/ArbitraryTokenPhasedSharesWrapperProxy.sol
```

### 2.1.1 Excluded from scope

Only the files mentioned above are in scope. Balancer V2, Aura Finance, Kiln and ParaSwap are not in scope and are expected to work correctly as documented.

## 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.

Avantgarde Finance adapted the Balancer V2 adapter to support native Balancer LP staking and implemented a new adapter and the surrounding infrastructure to allow staking Balancer LP tokens on Aura. Further, price feeds for the staked LP tokens and Balancer v2 stable pools LPs are introduced. The adapter for ParaSwap was extended to support the batching of orders where individual orders may fail. Additionally, an external position for staking ETH on Kiln is introduced. Last, Avantgarde Finance implemented periphery contracts that generalize the wrapping of fund shares so that can be bought using arbitrary deposit tokens.

### 2.2.1 Balancer v2 Stable Pool LP Price Feed

Avantgarde Finance implements the contract BalancerV2StablePoolPriceFeed which is a price feed for Balancer v2 stable pool LP tokens. The price feed allows governance to add and remove supported factories using `addPoolFactories()` and `removePoolFactories()`. These are used as a validation measure when governance adds pools with `addPools()`. Note that governance can unsupport pools through `removePools()`. Once a pool is added, `isSupportedAsset()` will return true. However, the price feed is only intended to be used with stable or meta stable pools.

Since, all assets in such pools are pegged to each other, the price will be approximated in terms of one underlying token, the proxy asset, which governance specifies when adding the pool. Hence, the value can be computed by multiplying the result of Balancer's `getRate()` function with the amount LP tokens and dividing it by the total supply. Note that the proxy asset shall be an underlying token of the pool or an asset that is pegged to them.

### 2.2.2 Balancer v2 LP staking

The Balancer v2 LP integration was adapted to support staking Balancer v2 LP tokens through Balancer or Aura. Note that these are separate adapters but share common functionality.

BalancerV2LiquidityAdapter was adapted and continues to support actions `lend()` and `redeem()`. The liquidity gauges of Balancer v2 are forks of Curve Finance's gauges. Hence, code has been reused and adapted.

Note that following actions have been added:

- `stake()`: Stake Balancer v2 LP tokens in a Balancer v2 liquidity gauge.
- `lendAndStake()`: Mint Balancer v2 LP tokens and stake them in a liquidity gauge, see `lend()` and `stake()`.
- `unstake()`: Unstake a Balancer v2 LP token from a liquidity gauge.
- `unstakeAndRedeem()`: Unstake Balancer v2 LP tokens from a liquidity gauge.
- `claimRewards()`: Claim rewards for a given LP token.

Additionally, a price feed that supports any asset where function `lp_token()` does not return `0x0` has been implemented so that it is priced in terms of the staked Balancer v2 LP token (same amounts).

Aura Finance allows to stake Curve LP tokens and BAL tokens to accrue additional rewards. Note that Aura Finance is a fork of Convex Finance. Hence, code has been reused and adapted. For Aura Finance, the wrapper for Convex positions is reused completely. The integration adapter is interacting with the ERC-20 wrappers for Aura and, unlike the Balancer staking adapter, does not support any operations that `lend` or `redeem` LPs. The following actions are supported:

- `stake()`: Stake a Balancer v2 LP token into the Aura pool.
- `unstake()`: Unstake a Balancer v2 LP token from the Aura pool.
- `claimRewards()`: Claim rewards for a given staking token (token representing the wrapped position).

Additionally, a price feed for the wrapped staked LPs on Aura has been implemented and is equivalent to the ConvexCurveLpStakingWrapperPriceFeed. Meaning, an asset is supported if it has been deployed by the Aura wrapper factory, and an asset's value is priced in terms of the staked Balancer v2 LP token (same amounts).

### 2.2.3 Kiln staking external position

Kiln is a staking platform where users can stake their ETH. Avantgarde Finance implements an external position that enables staking through Kiln.

The actions available are:

- `Stake`: Stakes multiples of 32 ETH on the Kiln staking contract.
- `ClaimFees`: Claims the accrued rewards. Note three different types of claiming are implemented. Type `ExecutionLayer` withdraws execution layer fees, type `ConsensusLayer` withdraws consensus layer fees, and type `All` withdraws both of them. Note that withdrawing consensus layer fees is not yet supported by Kiln.
- `WithdrawEth`: Withdraws any ETH balance the external position holds. Note that any address can claim the accrued fees on behalf of anyone. Hence, this function manages scenarios where someone has withdrawn fees on behalf of the external position.

Since no debt is taken, `getDebtAssets` will report no debt. The only managed asset will be ETH with a managed amount that equals the staked amount plus the current balance of the external position. Note that pending fees are not included in the position value.

### 2.2.4 ParaSwap V5 orders batching

The ParaSwap V5 integration was adapted to support batch orders through the `takeMultipleOrders` function. It is possible to specify whether the adapter should revert if one or more orders from the batch fail to be executed.

### 2.2.5 Shares Wrapper For Arbitrary Deposit Token

Avantgarde Finance introduces a new peripheral contract that wraps the buying and redeeming of shares to allow investing with arbitrary deposit tokens. Such wrapper contracts can be deployed through a factory with function `deploy()` that deploys a proxy pointing to the library. The wrapper implements the ERC-20 interface but transfers can be deactivated on initialization.

The wrapper has three phases:

1. Deposit phase: Users, potentially only whitelisted ones, can deposit the arbitrary deposit tokens with `deposit()` if the deposit limit is not exceeded. The wrapper shares are minted in a 1:1 ratio to the deposited asset. Further, users can withdraw their share of the underlying using `withdraw()`.
2. Locked phase: The fund owner or the wrapper manager can start the locked phase during the deposit phase using `enterLockedState()`. However, that requires a donation of the fund's

denomination asset to the wrapper contract so that shares can be minted. The deposit token is sent to the vault proxy afterwards.

3. Redeem phase: If the managed value of all external positions is zero, the fund owner or the wrapper manager can start the locked phase during the locked phase using `enterRedeemState()`. The fund shares are redeemed in kind so that the shares wrapper contract holds the assets. Users, now, can redeem the wrapped shares for the underlying tokens.

Note that two fees are collected:

- Protocol fee: Fee that is transferred to the protocol. The fee collection starts on the first deposit, ends when the redeem phase is entered. When the redeem phase is entered, the fees are collected.
- Local fee: Fee transferred to a fee recipient specified by the deployer. The fee can optionally exclude the principal amount of the deposit token.

Note that funds require specific setups for using such a wrapper (e.g. only wrapper can buy shares, core protocol fees are turned off, restrictions on asset managers). Further note, that maximum deposits can be bypassed through donations and that the maximum deposit and whitelist only prevent future deposits and depositors.

### 2.2.6 Trust Model

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

Governance is fully trusted and expected to not only behave honestly but also to fully understand the systems they are interacting with which includes choosing appropriate parameters. Fund managers and asset managers are generally fully trusted for a fund. However, their powers can be limited through the fund's settings.

All external systems are expected to be non-malicious and work correctly as documented. Further, for the Balancer v2 stable pool price feed we expect that, if the pool is a meta pool, the price provider contracts are not manipulatable.

We assume that the shares wrapper for arbitrary deposit tokens is setup correctly and is not setup with rebasing nor reentrant tokens.

In general we assume Enzyme only interacts with normal ERC-20 tokens that do not have multiple entrypoints nor callbacks.

# 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:

- 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 | 2 |

- Unexpected Staking of Tokens Risk Accepted
- Unhandled Stake Slashing on Kiln Risk Accepted

| | |
|---|---|
| Low-Severity Findings | 0 |

## 5.1 Unexpected Staking of Tokens

Correctness Medium Version 1 Risk Accepted

Since the spent assets are not validated against the Balancer v2 pool's underlying assets, `lendAndStake()` could stake LP tokens from the vault along with the newly generated ones.

Consider the following scenario

1. Vault holds 1 Balancer LP
2. Manager triggers `lendAndStake` where the underlyings of the Balancer LP's pool and Balancer LP are specified as spent assets.
3. 1 more Balancer LP is generated.
4. The full balance (2) of Balancer LPs is staked.

In contrast, all unused spent assets during the lending part, are returned to the vault proxy. Hence, the adapter may not behave as expected.

---

**Risk accepted:**

Avantgarde Finance replied:

```
This actually seems like an unintended
convenience (batches the staking of held LP tokens with buying +
staking new LP tokens). It is going to be the case for many/most
adapters that if the manager inputs an incorrect value, there could be
unintended consequences or value loss (e.g., slippage). Especially since
there is no reported path that leads to value loss here, we will leave as-
is.
```

## 5.2 Unhandled Stake Slashing on Kiln

Design | Medium | Version 1 | Risk Accepted

When computing the managed assets of an external position on Kiln, the system assumes the position holds `validatorCount * 32 ETH + address(this).balance`, thus not considering any stake slashing that may have occurred. This could lead to an over-evaluation of the position if the stake gets slashed on a validator.

---

**Risk accepted:**

Avantgarde Finance replied:

```
Rewards and slashing are not included in the
current position valuation, as this requires external oracle monitoring of
the consensus layer. The actual position value will deviate by some
percent from the ideal value, which will generally tend to be more and
more undervalued if we assume consensus rewards outweigh slashing
in most cases. For now, managers will need to be aware of this, and if
they require more precision, we can integrate a simple oracle to
monitor the delta.
```

# 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 |

- lendAndStake() May Interact With Two Pools and Leave Tokens Behind Code Corrected

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

- Event Emitted When Non-Existing Pool Is Removed Code Corrected
- Missing Sanitization for _feeBps Code Corrected
- Validation for Balancer Staking Code Corrected

## 6.1 `lendAndStake()` May Interact With Two Pools and Leave Tokens Behind

Correctness Medium Version 1 Code Corrected

`lendAndStake()` should mint LP tokens for a pool and stake them. However, it is possible that funds are deposited into one pool but another LP token is staked due to a potential mismatch between pool id and the staking token's LP token. Ultimately, the newly minted LP tokens are to be left in the adapter.

Consider the following scenario:

1. The vault holds LP token B.
2. A lend and stake action is started. The pool id is A, the staking token's underlying LP token is pool with id B. The spent assets are the underlying tokens of pool id A and the LP token B.
3. Through the lending, LP token A is received.
4. If the adapter's balance of LP token B (spent asset) is greater than its balance of LP token A, staking will be successful.
5. Only the spent assets are pushed back to the vault.
6. The minimum incoming checks can in the integration manager pass if the spent asset amount for LP token B is greater than the minimum incoming amount.

Ultimately, funds can be lost.

---

**Code corrected:**

The pool is now validated against the staking token's underlying BPT in `__parseAssetsForLendAndStake` and `__parseAssetsForUnstakeAndRedeem`.

## 6.2 Event Emitted When Non-Existing Pool Is Removed

Design Low Version 1 Code Corrected

Function `BalancerV2StablePoolPriceFeed.removePool` emits a `PoolRemoved` event even if a pool was never added. In contrast, function `BalancerV2StablePoolPriceFeed.removePoolFactories` emits events only if a previously added factory is removed.

---

**Code corrected:**

The check `isSupportedAsset(pool)` has been added to only allow the deletion, and emission of the associated event, of an existing pool.

## 6.3 Missing Sanitization for `_feeBps`

Design Low Version 1 Code Corrected

No input sanitization is done on `_feeBps` in function the `ArbitraryTokenPhasedSharesWrapperLib.init`. One could deploy with `feeBps > MAX_BPS` intentionally or by mistake, which would block the redemption because local fees cannot be paid out.

---

**Code corrected:**

Input sanitization for the `feeBps` has been added. It must satisfy `feeBps < MAX_BPS`.

## 6.4 Validation for Balancer Staking

Design Low Version 1 Code Corrected

Both, the Balancer native staking and the Aura staking, perform validity checks on the staking token when parsing the assets for staking or unstaking actions. The validation ensures that the LP token matches the staking token. That is typically implemented as follows:

```
__validateBptForStakingToken(stakingToken, __getBptForStakingToken(stakingToken));
```

However, that will perform the check that `__getBptForStakingToken(stakingToken)==__getBptForStakingToken(stakingToken)` which is always `true`. Hence, the validation is redundant and increases gas consumption.

---

**Code corrected:**

The redundant checks have been removed. However, no validation of the staking addresses for Balancer native staking tokens has 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 Unfair Distribution of Rebasing Tokens in Shares Wrapper

Note Version 1

In contrast to other system contracts, the shares wrapper for arbitrary deposit tokens does not support rebasing tokens.

For each deposit token wei deposited in the shares wrapper, one shares wrapper wei is minted to the depositor. If the deposit token is a rebasing token, that may lead to losses for early depositors in terms of rebase amounts.

Consider the following scenario:

1. Alice deposits 1 stETH and receives 1 stETH shares wrapper.
2. stETH rebases. The contract holds 2 stETH.
3. Bob deposits 1 stETH and receives 1 stETH shares wrapper.
4. Technically, Alice contributed to two thirds of the contracts holdings (2 stETH out of 3 stETH). However, Bob and Alice both have claims to 50% of the contract's underlyings.

Ultimately, early-depositors could lose rebase amounts.