PUBLIC

Code Assessment

of the Sulu Extensions VI Smart Contracts

October 13, 2022

Produced for

by

Contents

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

Avantgarde Finance implements changes for the external position for Compound to improve validation of the borrow and repay actions such that fund managers cannot mistakenly pay back zero amounts on unused cTokens which earlier removed the debt for a cToken with the same underlying. Additionally, changes were made to the fee reserve so that governance can do arbitrary calls from the fee reserve. Further, a new integration is implemented that allows minting and burning Balancer v2 LPs for arbitrary pools. For weighted Balancer v2 pools a pricefeed has been implemented. Last, a new external position type is introduced to integrate with Notional v2 so that depositing collateral, lending, borrowing, and paying back debt is possible.

The most critical subjects covered in our audit are functional correctness, interaction with external systems according to their documentation, and compatibility with the Enzyme system. Security regarding all the aforementioned subjects is high.

The general subjects covered are trustworthiness, documentation, and error handling. Security regarding all the aforementioned subjects is high.

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

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

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

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

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

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	3
• Code Corrected	1
• Specification Changed	1
• Acknowledged	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 Sulu Extensions VI 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	19 September 2022	07443093c77453d13c79482fc4bd77f9ab7f4d3e	Initial Version
2	10 October 2022	a3765347b7a1e770dd27fb5990efc5581181e239	Second Version

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

Compound Debt Position Changes:

```
* contracts/release/extensions/external-position-manager/external-positions/compound-debt/CompoundDebtPositionLib.sol
* contracts/release/extensions/external-position-manager/external-positions/compound-debt/CompoundDebtPositionParser.sol
* contracts/release/extensions/external-position-manager/external-positions/compound-debt/ICompoundDebtPosition.sol
```

ProtocolFeeReserve Changes:

```
* contracts/persistent/protocol-fee-reserve/ProtocolFeeReserveLib.sol
```

Balancer v2 LP Integration and LP Price Feed:

```
* contracts/release/extensions/integration-manager/integrations/adapters/BalancerV2LiquidityAdapter.sol
* contracts/release/extensions/integration-manager/integrations/utils/actions/BalancerV2ActionsMixin.sol
* contracts/release/infrastructure/price-feeds/derivatives/feeds/BalancerV2WeightedPoolPriceFeed.sol
* contracts/release/interfaces/IBalancerV2Vault.sol
* contracts/release/interfaces/IBalancerV2WeightedPool.sol
* contracts/release/interfaces/IBalancerV2WeightedPoolFactory.sol
```

Note that for the following contracts copied from Balancer we only compared whether the functions have the same semantics:

- contracts/release/utils/BalancerV2FixedPoint.sol
- contracts/release/utils/BalancerV2LogExpMath.sol

Notional Finance External Position:

```
* contracts/release/extensions/external-position-manager/external-positions/notional-v2/INotionalV2Position.sol
* contracts/release/extensions/external-position-manager/external-positions/notional-v2/NotionalV2PositionDataDecoder.sol
* contracts/release/extensions/external-position-manager/external-positions/notional-v2/NotionalV2PositionLib.sol
* contracts/release/extensions/external-position-manager/external-positions/notional-v2/NotionalV2PositionParser.sol
* contracts/release/interfaces/INotionalV2Router.sol
```

2.1.1 Excluded from scope

Only the files mentioned above are in scope. Compound is not in scope and is expected to work correctly as documented. Balancer v2 is not in scope and is expected to work correctly as documented. Notional v2 is not in scope and is 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 offers extensions, which expand the capability of Enzyme Sulu. Some extensions were changed, while others were newly added.

Please consider our previous audit report of the system for detailed descriptions of external positions and policies.

2.2.1 Compound Debt Position Changes

There was an issue with Compound debt positions, where for assets that have two cTokens (e.g. cwBTC and cwBTC2), it was possible to repay using a different one than was borrowed against. Since the debt against that cToken was zero, nothing was paid back. The enzyme system still considered the original debt repaid, which could lead to underreporting debt. This made the vault value higher than it should have been. The following changes have been made:

- For the RepayBorrow action, the CompoundDebtPositionLib now loads the cToken that was used to borrow the asset from storage. Hence, it is no longer possible to repay the debt for a different cToken than the one that was used to borrow, which would lead to debt becoming untracked.
- For the Borrow action, the parser now validates that a new borrow uses the same cToken for the underlying asset as previous borrows. If it is the only borrow of that asset, it stores the cToken. An asset can only be borrowed against one cToken at a time. If all debt is paid back, a new borrow can be made against a different cToken for the same asset.

2.2.2 Arbitrary Council Call on ProtocolFeeReserve Changes

The governance function `withdrawMlnTokenBalanceTo` is replaced with governance function `callOnContract`, which allows calling arbitrary contracts with arbitrary call data from ProtocolFeeReserve. The call is restricted to the Dispatcher Owner, which is the Enzyme council in the main deployment. The Enzyme Council is a fully trusted user in the trust model.

2.2.3 Balancer v2 LP Integration and LP Price Feed

Balancer v2 is an AMM that supports multiasset pools, on which users can swap tokens or provide liquidity. The core contract of the Balancer system is the vault, which is the central entrypoint for users. Pools are connected to the vault and are interacted with through hooks.

Avantgarde Finance provides a new integration for depositing into (mint pool LP tokens such that fees are earned) and for withdrawing (burn pool LP tokens) from Balancer v2. To support such assets, a new price feed for such LP tokens is offered.

The integration offers the following functionality to fund managers:

- `lend()`: Deposit underlying tokens to the vault, such that LP tokens for the desired pool are received, by calling `joinPool()` on the vault.
- `redeem()`: Exit from the vault, such that LP tokens are burnt and the underlying tokens of the pool are received, by calling `exitPool()` on the vault.

Note that holding LP tokens without a price feed or without a price feed for its underlying can lead to reverting price feeds.

The BalancerV2WeightedPoolPriceFeed provides the price of LP tokens based on the value of their underlying tokens. To be able to receive a price feed for a pool's LP tokens, the factory contract that deployed the pool must be added. Factories can be added and removed by governance through `addPoolFactories()` and `removePoolFactories()`. All pools deployed by a tracked factory will be supported assets (`isSupportedAsset()` returns `true`). However, the price feed is only intended to be used with weighted pools.

The price of LP tokens is computed as

$$Price_{LP} = \frac{\prod_i Balance_i^{weight_i}}{totalSupply_{LP}} \prod_i Price_i^{weight_i}$$

where the weights, total supply and balances are queried from Balancer v2, and the prices are the token prices in an intermediary asset. Please see Balancer's documentation.

2.2.4 Notional Finance integration

Notional Finance is a protocol for fixed rate lending and borrowing.

Avantgarde Finance has created a new external position type, which can hold assets in Notional Finance. Note that it integrates with the fCash and regular cash mechanics of Notional.

The external position offers the following actions to fund managers:

- `AddCollateral` deposits assets into Notional finance, where they are stored as cTokens. These can be used as collateral but will not be lent on Notional. It calls the Notional router's `depositUnderlyingToken()` function.
- `Lend` lends assets at a fixed term on Notional, which is represented as a positive fCash balance. It calls Notional router's `batchBalanceAndTradeAction` function. fCash can is also considered to be collateral on Notional. fCash cannot be redeemed until the maturity date has passed.
- `Redeem` redeems yieldTokens from Notional after lent fCash reaches maturity. It calls Notional router's `withdraw()` function.
- `Borrow` borrows assets with a fixed term from Notional, against cToken or fCash collateral. Borrowed assets are represented as a negative fCash balance. It calls Notional router's `batchBalanceAndTradeAction()` function.

The NotionalV2PositionLib also offers external functions for calculating the value of the external position. `getManagedAssets()` returns all positively valued assets and their amounts, `getDebtAssets()` returns all negatively valued assets (debts) and their amounts. The amount of underlying tokens represented by fCash is approximated using the Notional router's `getPresentfCashValue()`, which values the fCash based on their value at redemption, the current interest rate and the time to maturity.

2.2.5 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.

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

2.2.6 Changes in V2

The Compound position parser now solely validates against the price feed while the library now validates against the stored cToken.

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	1

5.1 Temporary DOS Through Donations

Design Low Version 1 Acknowledged

In Notional, depositing collateral for others is possible. For example, `depositUnderlyingToken` can deposit collateral to another address than `msg.sender`. Hence, it is possible to donate collateral to a position in such a way that it becomes tracked within the Notional system. Since the external position computes the managed assets based on what Notional's `getAccount` returns, such donations will become visible to the external position. Hence, it could be possible to temporarily DOS the position by donating to it an unsupported token.

Acknowledged:

Avantgarde Finance replied:

```
Preventative measures for this are challenging and add complexity, so since the worst case is that the position will have a reverting price,
and since the owner can resolve this state by removing that collateral, we will provide a fix if this ever becomes an issue in practice.
```

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
Medium-Severity Findings	1
Low-Severity Findings	2

- Balancer Price Feed Vulnerable to Read-Only Reentrancy Code Corrected

- Borrowing From cTokens With Same Underlying Can Lead to Unreported Debt Code Corrected

- Remaining BPT in Adapter Code Corrected
- mulUp Incorrect Comment Specification Changed

6.1 Balancer Price Feed Vulnerable to Read-Only Reentrancy

Security High Version 1 Code Corrected

Balancer's system is vulnerable to read-only reentrancy. During the removal of liquidity, an inconsistency between the total supply and a pool's balances can be created (using native ETH transfers). That can be leveraged to manipulate the price feed upwards - leading to an over-evaluation of the fund.

Code corrected:

Now, a reentrancy protected call to `setRelayerApproval()` is made when the price is computed to ensure that Balancer is not reentered.

6.2 Borrowing From cTokens With Same Underlying Can Lead to Unreported Debt

Correctness Medium Version 1 Code Corrected

Some cTokens may have the same underlying (e.g. cWBTC and cWBTC2). The parser validates the cTokens to borrow from as follows:

```
// validate ctokens
for (uint256 i; i < cTokens.length; i++) {
    address cTokenStored = ICompoundDebtPosition(_externalPosition)
        .getCTokenFromBorrowedAsset(assets[i]);

    if (cTokenStored == address(0)) {
```

```
        require(
            CompoundPriceFeed(getCompoundPriceFeed()).getTokenFromCToken(cTokens[i]) ==
                assets[i],
            "parseAssetsForAction: Bad token cToken pair"
        );
    } else {
        require(
            cTokenStored == cTokens[i],
            "parseAssetsForAction: Assets can only be borrowed from one cToken"
        );
    }
}
```

Note that the validation aims to prohibit borrowing from two cTokens that have the same underlying. In most cases, this works correctly. However, borrowing from both cTokens (with the same underlying) for the first time in the same action will bypass the validation. Consider the following scenario:

1. Borrow for the first time from both cWBTC and cWBTC2.
2. In the first iteration of the loop, `cTokenStored` will be `0x0` due to WBTC never being borrowed.
3. In the second iteration of the loop, `cTokenStored` will still be `0x0` since the mapping in the external position has not been updated yet. The update will happen in `__borrowAssets`, after the parser returns.

This will allow the external position to borrow from both cTokens. Note that `__borrowAssets` will only keep track of the first cToken. Hence, debt of the second cToken will not be tracked. The total debt will be underreported in such a scenario.

Code corrected:

The parser now solely validates against the price feed while the library now validates against the stored cToken.

6.3 Remaining BPT in Adapter

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

Avantgarde Finance reported an issue when redeeming Balancer LP tokens. It was possible to redeem BPTs so that a maximum amount of burned LP tokens is specified along with exact received underlying amounts. If the maximum was not reached, the BPT remained in the adapter.

Code corrected:

After redemption, any surplus BPT remaining in the contract is sent back to the vault proxy.

6.4 mulUp Incorrect Comment

Correctness **Low** **Version 1** **Specification Changed**

```
function mulUp(uint256 _a, uint256 _b) internal pure returns (uint256 res_) {
   uint256 product = _a * _b;
   require(_a == 0 || product / _a == _b, "mul overflow");
```

```
    if (product == 0) {
        return 0;
    } else {
        // The traditional divUp formula is:
        // divUp(x, y) := (x + y - 1) / y
        // To avoid intermediate overflow in the addition, we distribute the division and get:
        // divUp(x, y) := (x - 1) / y + 1
        // Note that this requires x != 0, which we already tested for.

        return ((product - 1) / ONE) + 1;
    }
}
```

The comment in the `mulUp()` function of BalancerV2FixedPoint mentions divUp. It was likely copied from there and not changed. This issue is also present in the Balancer contract that `mulUp()` was adopted from.

Specification changed:

The comments in the files were adapted to reflect that the comments are not reviewed.