PUBLIC

Code Assessment

of the Sulu Extensions XIV Smart Contracts

December 21, 2023

Produced for

by

Contents

1 Executive Summary

Dear Enzyme team,

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

In the development of version 5 of the protocol, which builds on the existing Sulu system, Avantgarde Finance has implemented several changes to the connectors for external systems. The ZeroEx Adapter now includes support for over-the-counter (OTC) orders. Additionally, a new external position has been introduced to interact with Term Finance, supporting lending only. For the existing Maple integration, outdated code related to version 1 and its corresponding migration code have been removed. Furthermore, the validation function has been updated to reflect recent changes in Maple.

The most critical subjects covered in our audit are asset solvency, functional correctness, front-running, and accurate fund valuation. The security of all aforementioned subjects is high. Please note that there might be some unexpected scenarios (e.g. undercollateralized loans in Term Finance) that are intentionally unhandled, see System Overview, Assessment Overview and Notes.

The general subjects covered are code complexity, upgradeability, unit testing, and documentation. The security of all aforementioned subjects is high. However, note that in some scenarios the system may fail to untrack positions which could lead to increased gas costs, see Failing to untrack offers of cancelled auctions.

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	1
• Code Corrected	1
Low-Severity Findings	2
• Code Corrected	1
• Risk Accepted	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 XIV 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	5 December 2023	70678cbb87592e8ac7875511712e4f75a81bda42	Initial Version
2	20 December 2023	69dc115e5d4502bdf85973582cf5408d15b6bfe4	After intermediate report

For the solidity smart contracts, the compiler version `0.6.12` or `0.8.19` respectively was chosen.

The smart contracts in scope are:

2.1.1 ZeroExv4: Supporting otc order type

`release/extensions/integration-manager/integrations`

- `adapters/ZeroExV4Adapter.sol`
- `utils/0.6.12/actions/ZeroExV4ActionsMixin.sol`

2.1.2 External Position Term Finance

`release/extensions/external-position-manager/external-positions/term-finance-v1-lending`

- `TermFinanceV1LendingPositionDataDecoder.sol`
- `TermFinanceV1LendingPositionLib.sol`
- `TermFinanceV1LendingPositionParser.sol`
- `bases/TermFinanceV1LendingPositionLibBase1.sol`

2.1.3 External Position Maple: update validation function

`contracts/release/extensions/external-position-manager/external-positions/maple-liquidity`

- `MapleLiquidityPositionParser.sol`

2.1.4 External Position Maple: remove old v1 and migration code

`contracts/release/extensions/external-position-manager/external-positions/maple-liquidity`

- `MapleLiquidityPositionDataDecoder.sol`
- `MapleLiquidityPositionLib.sol`
- `MapleLiquidityPositionParser.sol`
- `MapleLiquidityPositionLibBase1.sol`

2.1.5 Excluded from Scope

All the contracts that are not explicitly mentioned in the Scope section are excluded from scope. The external systems, with which the Sulu interacts, are assumed to work correctly. Moreover, all the libraries used such as the OpenZeppelin library are assumed to work correctly and not in scope of this review. Finally, attack vectors, such as unfavourable for the fund trades, employed by the managers of the funds have not been considered as the managers are considered trusted by the system.

Changes in the core contracts have not been reviewed in this scope, as they are still a work in progress (WIP). These will be reviewed at a later point in time.

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.

This assessment is only concerned with specific code parts of the system. Here, we only provide a brief overview of the newly introduced modules of the system. Please refer to Enzyme's documentation or our previous reports for a more detailed overview of the system.

Enzyme is an investment management system that allows one to buy shares of certain vaults. Vaults act like funds and each vault has a vault manager, who is in charge of investing the funds of the vault into other DeFi protocols. The manager can set certain policies that limit the actions of the users including the manager's. Furthermore, the manager can only invest in third-party protocols for which an adapter exists. Avantgarde Finance expands the system by adding new policies, adapters, and external positions to expand and improve the investment opportunities for the vault managers.

The assessment was performed on the following new or refactored functionalities which integrate into Enzyme Finance's overall ecosystem:

- Added support for OTC orders in the existing ZeroExV4Adapter.
- A new integration for Term Finance (lending only)
- A minor update has been made to the Maple integration to reflect recent changes in Maple.
- Obsolete Maple version 1 and its respective migration support have been removed.

The system overview is divided into multiple sections that will provide a brief description of each of the code components above.

2.2.1 Added support for OTC orders in the existing ZeroExV4Adapter

Support for OTC orders has been added to the existing ZeroExV4Adapter. The Adapter now facilitates OTC order types, which are fullfilled using `zero_ex_v4_exchange.fillotc()`. Callers provide an order structure along with the corresponding signature.

2.2.2 Term Finance (lending only)

The Term Finance Protocol v1 facilitates noncustodial, fixed-rate lending on-chain through an auction process that matches borrowers and lenders based on sealed bids and offers. Borrowers receive loans and lenders obtain ERC-20 Term Repo Tokens, with smart contracts managing repayments and collateral.

Please consider the relevant parts of Term Finaces Documentation for further details https://docs.term.finance/

The external position implements the following actions:

- `AddOrUpdateOffers` Allow to add or updated (increase or decrease) an offer for an auction Funds are retrieved from / returned to the vault.
- `RemoveOffers`: Allows to remove an offer from an ongoing auction. Received tokens are returned to the vault.
- `Redeem`: Allows redemption of loan tokens for the underlying (purchase tokens). Received tokens are returned to the vault.
- `Sweep`: The function iterates through all purchase tokens of registered TermAuctions. If the ExternalPosition has a non-zero balance of any token, it returns these funds to the vault. This is typically used when an offer remains unfulfilled or when purchase tokens have been returned to the External Position for other reasons.

`getManagedAssets()` returns the value of managed assets which can be held in four different ways: 1. Underlying tokens refunded to the external position, as defined by the purchase token of the respective auction. 2. Outstanding offers inopen auctions. 3. Repo tokens representing loans that have not yet reached maturity. 4. Repo tokens for loans that have reached maturity and are redeemable.

Warning: There is a risk of loans being undercollateralized and losing value. This is not supported and hence the value returned may be incorrect. Such scenarios are generally unlikely to occur but can lead to wrong valuation of a fund.

This position has no debt assets, hence `getDebtAssets()` returns empty values.

2.2.3 Maple

Fix: Due to Maple's recent update to their Global contract, the isFactory() function is no longer callable. The code has been revised to use the new isInstanceOf function for verifying interactions with authentic Maple contracts.

Clean-up: All code related to Maple version 1 and the migration from Maple v1 to v2 has been removed. The updated code now exclusively supports interactions with Maple version 2.

2.2.4 Roles and Trust Model

Please refer to the main audit report and the extension audit reports for a general trust model of Sulu. Note that all external systems are expected to be non-malicious and work correctly as documented.

TermFinance is fully trusted and assumed to work as documented. We assume that the priviledged roles of TermFinance act responsibly and in a timely manner. For example, we expect that they complete/cancel auctions timely manner. Further, note that TermFinance contracts are upgradable,

hence may change their behavior. Also, fund managers are expected to reveal the submitted hashed prices.

No new roles are introduced by the components under review.

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

- Failing to Untrack Offers of Cancelled Auctions Risk Accepted

5.1 Failing to Untrack Offers of Cancelled Auctions

Design Low Version 1 Risk Accepted

CS-SUL14-001

`RemoveOffers` can untrack offers when the auction has not started yet or when it has been cancelled for withdrawal. `Redeem` can only untrack offers when repo tokens can be redeemed and at least one repo token is in the contract. `Sweep` can only untrack offers if the auction has completed and if the balance of repo tokens is zero.

If the auction has been cancelled, offers cannot be untracked. While action `sweep` can be used to return the purchase tokens to the vault, it does not untrack the offer. Ultimately, that may lead to increased gas consumption.

Risk accepted:

Avantgarde Finance replied:

```
After consultation with the Term Finance team, this risk is accepted since (1) there
is no way to observe cancelled auctions on-chain, (2) cancellations are rare
(has not occurred on mainnet yet), and (3) the worst case is gas cost bloat
(i.e., not security or correctness)
```

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	1
Informational Findings	1

- Double Tracking Term Contracts, Inflation of EP Valuation Code Corrected
- Term EP Incorrect Loan Value Code Corrected
- Delta Is Not Applied on Current State Code Corrected
- Inconsistent Array Lengths for Term Finance EP Code Corrected

6.1 Double Tracking Term Contracts, Inflation of EP Valuation

Security High Version 1 Code Corrected

CS-SUL14-005

The `AddOrUpdateOffers` action in the Term Finance EP is responsible for publishing new offers or updating existing ones. When this action is called for a term auction without any tracked offers and with an empty "submittedOfferIds" parameter (represented as empty bytes32[]), no checks are done, the term auction is simply added to the array `termAuctions`. This process can be repeated multiple times, essentially pushing the same term auction address into the `termAuctions` array on each call.

Afterwards actual offers can be added for this `termAuction`.

`getManagedAssets()` iterates through the array of tracked "termAuctions" and assesses the individual offers. It depends on the assumption that the "termAuction" contract exists in the array only once. Due to the repetitive presence of the same "termAuction" address in the array, the same offers are counted multiple times. As a result, this leads to an inaccurate and inflated valuation for the external position.

This could be abused by a malicious manager to inflate the positions valuation.

Code corrected:

It is now ensured that the submitted offer IDs is not empty. As a consequence, at least one ID is published to Term Finance. Hence, no term contract can be added without having at least one offer ID.

6.2 Term EP Incorrect Loan Value

Correctness | Medium | Version 1 | Code Corrected

CS-SUL14-002

In `getManagedAssets()`, `__getLoanValue()` is used to calculate the value of a loan. Before maturity of a loan, the valuation of the offers may be inaccurate. Namely, a multiplication with `redemptionValue()` is missing. Hence, the repo tokens are treated as 1:1 to the underlying during the lifespan of a loan. Once expiry has been reached, `redemptionValue()` is taken into account (this is in line with what the tokens may be redeemed for).

As a consequence, the value per offer will evolve as follows:

1. Before auction completion: `offer.amount`.
2. Right after auction completion `offer.amount / redemptionValue` (assuming the auction completion happened right at the term start. Note that value is the combination of the formulas of how many repo tokens Term Finance mints and how Enzyme valuates them - resulting in a function based on the offer's amount).
3. Linear increase for the principal amount computed in 2. over time according to interest rate (`offer.amount/redemptionValue*(1+clearingPrice*timePassedSoFar)` - again this formula is a combination of formulas to result in a formula based on the initial offered amount). Right before the end of the term, the value should be roughly what 4. is as nearly no time needs to pass.
4. `offer.amount * (1+clearingPrice*totalTime)`

To summarize, during the life time the loan value is not scaled by `redemptionValue()` which will lead to an undervaluation of the position. It is best visible at the borders of the lifespan of a loan where the value suddenly jumps to lower/higher values.

Code corrected:

The code now multiplies with `redemptionValue()` where the multiplication was missing.

6.3 Delta Is Not Applied on Current State

Design | Low | Version 1 | Code Corrected

CS-SUL14-003

When adding an offer or updating it's amount, the delta may be applied to outdated values, leading to potentially undetected decreases. Consider the following example:

1. An offer submission is created with a delta of 100 (assume the offer ID will be X). That will lead to an offered amount of 100 for the submission.
2. The offer at X is updated with an offer submission with a delta of 50. Since the submission of 1. has not been published, this computation will work on "outdated" values. Namely, this will lead to an offered amount of 50.
3. The changes applied on the corresponding Term contract. Ultimately, one offer will be present with X having 50 tokens offered.
4. As a consequence, 50 tokens are left in the contract and not swept (no decrease detected).

Note that tokens could unnecessarily be left in the EP.

Code corrected:

The code has been adjusted. In case the offer IDs returned by the term auciton offer locker contract are not unique in the array, the execution reverts. Hence, the scenario of modifying two times the same offer ID in one go is not allowed anymore.

6.4 Inconsistent Array Lengths for Term Finance EP

Informational | Version 1 | Code Corrected

CS-SUL14-004

The `_actionArgs` of action `AddOrUpdateOffers` consists of three arrays that should have the same length. However, that is not enforced and, hence, unexpected behaviour could occur. Namely, if the amounts have a length greater than the offer IDs, the parser will compute a too high amount of funds to be transferred to the EP. The price hashes could also have a length greater than the offer IDs which results in some price hashes not being used (unspecified behaviour).

Code corrected:

It is now validated that the arrays have the same length.

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 Jump in Value at Auction Completion

Note Version 1

An auction could be completed after the term start. As a consequence, once the auction is completed, the value of the EP may jump from the constant value of the offered amounts to a value that includes some interest.

7.2 TermFinance: Permissionless Redemption

Note Version 1

After maturity, anyone can trigger the redemption of repo tokens by calling `TermRepoServicer.redeemTermRepoTokens()`. Resulting int the external position receiving the purchase token. Using action `Sweep`, these tokens can be returned to the vault.

7.3 ZeroEx V4 OTC Order Type Incompatible With GasRelayer

Note Version 1

In ZeroEx v4, OTC orders include a field `tx.origin` which must match the actual `tx.origin` of the execution. This verification ensures that only the authorized party can execute the order. Executing an order through the adapter, this is typically the fund manager (the caller of callOnExtension).

Consequently the GasRelayer is unusable in this scenario.