PUBLIC

Code Assessment

of the Sulu Extensions II Smart Contracts

March 4, 2022

Produced for

enzyme

by

CHAINSECURITY

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

This extensions introduces small changes in the Sulu core (Protocol Fee: Conditional burn or transfer of $MLN, Position Limit is now an immutable, Overhead handling finalization of Synthetix has been removed, 1st action after creating an ExternalPosition). Additionally a simplified PerformanceFee, and a ConvexCurvePool staking wrapper have been reviewed.

During the review, no important issues were uncovered. The most critical subjects covered in our audit are functional correctness, access control and precision of arithmetic operations. Security regarding all the aforementioned subjects is high. General subjects covered were code complexity, gas efficiency, documentation and specification. All the aforementioned subjects were of high quality.

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 and the issues uncovered.

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	0
Low-Severity Findings	2
• Code Corrected	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 II 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	31 January 2022	184aa4facbfdb49c30065370324febb01f227271	Initial Version
2	9 Febrary 2022	a5de1294a55589bfec39c85c1e192e09a2f3800d	Second Version
3	3 March 2022	1dca776e53254289b5b40ba57173549bce47e3e2	Final Version

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

For the changes in the core protocol of the Sulu release, the changes in following files (compared to the main Chainsecurity audit of the Sulu release and extensions thereof) have been reviewed:

contracts/persistent/protocol-fee-reserve/ProtocolFeeReserveLib.sol
contracts/persistent/protocol-fee-reserve/bases/ProtocolFeeReserveLibBase1.sol
contracts/release/core/fund/vault/VaultLib.sol contracts/release/core/fund/comptroller/ComptrollerLib.sol
contracts/release/core/fund/comptroller/IComptroller.sol
contracts/release/extensions/fee-manager/FeeManager.sol
contracts/release/extensions/fee-manager/fees/PerformanceFee.sol
contracts/release/extensions/integration-manager/integrations/adapters/SynthetixAdapter.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/SynthetixActionsMixin.sol
contracts/release/extensions/external-position-manager/ExternalPositionManager.sol

For the new functionality regarding the staking wrapper and the convex curve pool staking , the following files have been reviewed:

contracts/release/infrastructure/staking-wrappers/IStakingWrapper.sol
contracts/release/infrastructure/staking-wrappers/StakingWrapperBase.sol
contracts/release/infrastructure/staking-wrappers/StakingWrapperLibBase.sol contracts/release/infrastructure/staking-wrappers/convex-curve-lp/ConvexCurveLpStakingWrapperFactory.sol contracts/release/infrastructure/staking-wrappers/convex-curve-lp/ConvexCurveLpStakingWrapperLib.sol
contracts/release/interfaces/IConvexBaseRewardPool.sol
contracts/release/interfaces/IConvexBooster.sol
contracts/release/interfaces/IConvexVirtualBalanceRewardPool.sol
contracts/release/extensions/integration-manager/integrations/utils/bases/CurveLiquidityAdapterBase.sol
contracts/release/extensions/integration-manager/integrations/adapters/ConvexCurveLpStakingAdapter.sol contracts/release/extensions/integration-manager/integrations/utils/actions/StakingWrapperActionsMixin.sol contracts/release/infrastructure/price-feeds/derivatives/feeds/ConvexCurveLpStakingWrapperPriceFeed.sol

and the corresponding changes in:

contracts/release/utils/beacon-proxy/BeaconProxyFactory.sol
contracts/release/extensions/integration-manager/integrations/adapters/CurveLiquidityAaveAdapter.sol
contracts/release/extensions/integration-manager/integrations/adapters/CurveLiquidityAdapter.sol
contracts/release/extensions/integration-manager/integrations/adapters/CurveLiquiditySethAdapter.sol
contracts/release/extensions/integration-manager/integrations/adapters/CurveLiquidityStethAdapter.sol c

ontracts/release/extensions/integration-manager/integrations/utils/actions/CurveGaugeV2RewardsHandlerMixin.sol

For details regarding the changes in scope please refer to the system overview.

2.1.1 Excluded from scope

The rest of the contracts.

The external systems, notably Convex Finance and Curve.Finance are not part of this review and expected to work correctly as documented.

For the StakingWrapper, no specification for the harvest functionality, the calculation of claimable & unaccounted rewards and the checkpoint functionality has been provided. The review for functional correctness hence was limited to best effort only.

2.2 System Overview

This system overview describes the changed and new functionality reviewed as part of this report. For the main system description and trust model, please refer to the main audit of the Sulu release.

Changes in the core of the Sulu release

1. Conditional burn or transfer of $MLN upon protocol fee shares buyback

MLN tokens are native to the Ethereum mainnet and can only be burned on this network. The functionality to conditionally collect and transfer the $MLN received by the protocol as payment for the protocol fee has been added to allow deployment of the Enzyme protocol on other networks. The Council can decide whether to assign a trusted party to bridge and burn the $MLN on Ethereum, or to leave the collected $MLN dormant for the mid-term solution.

2. Change VaultLib.POSITIONS_LIMIT to an immutable var

The previous constant was changed to an immutable which allows to deploy the same code on all networks and specify the limit upon deployment.

3. Remove Synthetix considerations from Sulu core

Due to the special behavior around the finality of balances of synths the current version of the Enzyme Protocol contains extensive logic. This logic to ascertain the finality introduces a significant overhead in terms of gas. Lately, due to the rise of L2 solutions and the expansion of Synthetix to these other networks, the synthetix asset universe on mainnet has been reduced. Starting with Sulu, only sUSD will remain in the asset universe.

The special behavior enforcing the finality of synths has been removed from the core protocol logic.

Funds are encouraged to trade out any synths other than sUSD before migration to the new Sulu release. Note that after a successful migration to Sulu it's still possible to trade out other synths via an adapter the policies of this funds allow to use. The Synthetix adapter has been simplified to only exchange of arbitrary synths into sUSD.

4. Allow a 1st action when creating new ExternalPositionProxy instances

This is a gas optimization which allows to specify a first action to execute after the creation of a new external position.

Simplified PerformanceFee

This fee based on the performance of the share price takes a percentage of the profit the fund made.

The previous PerformanceFee was quite complex and included a crystallization period before the performance fee could actually be withdrawn. The implementation of this was quite complex and had some unexpected behavior.

This new simplified version distributes the fee whenever the share price reaches a new high. The increase of the share price since the last settlement is taken as base for the payout calculation. This settlement is not only triggered automatically before shares are bought/sold but can also be manually triggered at any time. Upon settlement the fee is paid out by minting new shares (dilution of the shares value). After all other fees have been settled, the fee updates by recording the current share price as highest share price, which is the base line for the next settlement of this fee.

Convex Curve pool staking

The current version of the Enzyme Protocol already supports interaction with Curve.finance through various adapters for the different pools.

Convex Finance allows to stake Curve LP tokens and CRV tokens to accrue additional rewards.

A new adapter allowing funds to interact with Convex Finance has been introduced. As positions in Convex Finance are not represented by an ERC20 tokens, a wrapper has been created. This wrapper is separate from the Enzyme Protocol and works like the Curve gauge wrappers. Anyone may use it to deposit into a convex pool and receive ERC20 tokens representing their position. This ERC20 position can be held by funds in Enzyme.

The wrapper contracts issues wrapped convex position tokens in a 1:1 ratio. Hence, it requires checkpointing on every balance change to ensure fair reward sharing among token holders. The reward amount a user can claim per each of his wrapper tokens can be computed as a difference of two integrals, the first being the sum of all harvest amounts divided by the supply at the harvest moment, and the second being the value of the first integral when the user had his last balance change.

A global and user specific integrals, which describe the global rewards per tare required.

It computes the reward token balance differences and defines an integral which is the sum of reward tokens per wrapper token.

The new adapter making use of this wrapper features the following functionality:

- claimRewards: Allows the fund to claim rewards for a given staking token (token representing the wrapped position)
- lendAndStake: Deposit into a a standard curve pool for LP tokens and staking of these LP tokens into the convex pool
- stake: Stake LP tokens into the convex pool
- unstake: Unstake LP tokens from the convex pool
- unstakeAndRedeem: Unstake the LP tokens from the convex pool and redeem the LP tokens for a/the underlying depending on the parameters

A pricefeed is introduced for the wrapped convex positions. Since the convex position wrapper token and the curve LP token are exchangeable 1:1, the wrapper pricefeed will direct the price definition to the underlying curve LP pricefeed.

2.2.1 Trust Model

The trust model is defined as in previous reports. However, note that we assume the reward tokens of the convex pools to be a trusted default ERC-20 tokens without special behaviour (e.g. fees on transfer, transfer hooks to the receiver, rebasing tokens, ...)

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

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

- Gas Inefficiencies Acknowledged

5.1 Gas Inefficiencies

Design Low Version 1 Acknowledged

`addExtraRewards` iterates over all extra reward tokens to check whether the `rewardTokens` array contains them. However, the `rewardTokens` array is loaded from storage on every iteration. The gas consumption could be reduced by caching the array into memory or using a set like data structure for checking whether a token is already present.

Acknowledged:

Avantgarde Finance replied:

```
We attempted implementing the suggested
optimization, but rather than leading to savings, it led to inefficiencies
in the most frequent case and a more complex code surface area. The
case where Convex extra pool tokens are >1 is extremely rare (the vast
majority of Curve pools have 0 or 1 extra rewards tokens), and the extra
logic involved with copying `rewardTokens` into memory, validating
that it is a unique set, etc makes the refactor more expensive rather
than less in the vast majority of cases. For those rare cases, since
`rewardTokens` is already accessed in the first loop, all SLOAD
operations are already warm lookups, so the gas hit isn't significant.
```

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	0
Low-Severity Findings	1

- Potential Reentrancy Code Corrected

6.1 Potential Reentrancy

Security Low Version 1 Code Corrected

Transfers modify balances of users. Hence, checkpointing is required to be performed before any balances modification to ensure fair distribution of rewards. However, transfer functions are not reentrancy-protected which offers the following attack vector:

1. Assume a Convex pool staking wrapper contract where one user holds 50 out of 100 tokens while no rewards have been earned so far. Also assume that one reward token has an on-receive-hook to the recipient of the token.
2. Now, the attacker contract calls `claimRewardsFor()` to send rewards to itself. `___checkpointAndClaim` is called internally which harvests the Convex pool and then proceeds to checkpointing and claiming with `__updateHarvestAndClaim`.
3. 100 reward tokens are harvested. The total and the user integral are updated accordingly. The amount to transfer to the attacking contract is 50 reward tokens. However, the claimable amount is set to 0 in storage due to the transfer. Note, that `lastCheckpointBalance` is not updated.
4. **The transfer starts and modifies the balance to 50 and then calls the attacking contracts hook.**
    1. The attacking contract reenters the wrapper contract in the reentereable `transfer()` function inherited from ERC20.
    2. `___checkpoint` is called. Harvesting Convex has no effect but now `__updateHarvest` is called.
    3. The last checkpointed balance (still 0) and the current balance (now 50) are queried. The difference implies more rewards.
    4. Now, the integrals are updated and so is the claimable amount is now set to 25 for the attacking contract. The checkpointed balance is now set to 150.
5. The execution returns and nothing happens since the checkpointed balance is equal to the balance. No event is emitted.
6. The attacking contract claims his claimable amount. Totally, the attacking contract has claimed 75 instead of 50 reward tokens.

Ultimately, accounting issues occur since there are less rewards available than expected. Also, some user will potentially not be able to withdraw their LP tokens due to impossible transfers.

Even though we specify reward tokens to be regular ERC-20 tokens, it could be possible that, since the future is unforeseeable, ERC-777 tokens could be added as rewards, which would open up such attack vectors. Hence, the underlying issue is that the Checks-Effect-Interaction design pattern is not followed.

Code corrected:

The `nonReentrant` modifier was added to `_transfer`. Hence, all entrypoints that perform checkpointing are protected from reentrancy. Additionally, all checkpointing variables were made `private`. Hence, more derived contracts are protected from reentrancy attack vectors modifying checkpointing state.