PUBLIC

Code Assessment

of the Sulu Extensions V Smart Contracts

August 15, 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 V according to Scope to support you in forming an opinion on their security risks.

Avantgarde Finance implements two new policies that allow fine-grained access control on adapters and external positions and a new list registry for unsigned integers used by the latter policy. Additionally, a derivative price feed for FIDU, an LP token with USDC as its underlying, is introduced. Further, a manual value oracle is implemented that allows its owner to set arbitrary `uint256` values while keeping track of the latest update time. For its ownership transfers a new mixin is offered that implements the ownership transfer and claim mechanism. Arbitrary uncollateralized loans are offered through a new type of external position that allows to plug in accounting modules that compute the interest owed. Two such accounting modules are offered where one leverages the manual value oracle and the second one implements fixed interest. Two new external position types are also introduced to integrate with Solv Protocol's convertible vouchers from the buyer and from the issuer side. Lastly, Avantgarde Finance updated the DepositWrapper contract.

The most critical subjects covered in our audit are functional correctness, access control and compatibility with the Enzyme system.

Security regarding all subjects is high.

The general subjects covered are error trustworthiness, documentation, and interaction with external systems according to their documentation. Compatibility with external systems is extensive. However, note that for compatibility with Solv requires an upgrade by Solv, see Solv's BUYER_PAY fee pay type is unsupported is valid. Documentation is good. Trustworthiness is high given the trust model. However, please consider the note Arbitrary Loan Powers.

In summary, we find that the codebase provides an improvable level of security. Note that most items covered are of high 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	5
• Code Corrected	4
• Specification Changed	1
Low-Severity Findings	3
• Code Corrected	2
• Code Partially Corrected	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 V 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	27 June 2022	e40831fa183e9cfa80ef819b7249ec13f51c2eef	Initial Version
2	04 July 2022	8497bd50fb72a0cf594e3d60520cb5030d4048c6	Full Scope
3	20 July 2022	86dd0ee1b982dba73d3e9c0174de60c407ab9043	After intermediate report
4	08 August 2022	defe66dc1382fff066e0e0b8b1d996e50115e317	Arbitrary Loan Fix
5	12 August 2022	15c44609740efec7436d515b0cfa69d836e45d30	Final fixes and prettier

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

UintListRegistry:

```
* contracts/persistent/uint-list-registry/UintListRegistry.sol
```

Uint256ArrayLib:

```
* contracts/release/utils/Uint256ArrayLib.sol
```

Policies:

```
* contracts/release/extensions/policy-manager/policies/asset-managers/AllowedAdaptersPerManagerPolicy.sol
* contracts/release/extensions/policy-manager/policies/asset-managers/AllowedExternalPositionTypesPerManagerPolicy.sol
* contracts/release/extensions/policy-manager/policies/utils/AddressListRegistryPerUserPolicyBase.sol
* contracts/release/extensions/policy-manager/policies/utils/UintListRegistryPerUserPolicyBase.sol
```

Price Feed: FIDU:

```
* contracts/release/infrastructure/price-feeds/derivatives/feeds/FiduPriceFeed.sol
* contracts/release/interfaces/IGoldfinchConfig.sol
* contracts/release/interfaces/IGoldfinchSeniorPool.sol
```

Nominated Owner Mixin:

```
* contracts/release/utils/NominatedOwnerMixin.sol
```

Manual Value Oracle:

```
* contracts/persistent/arbitrary-value-oracles/IArbitraryValueOracle.sol
* contracts/persistent/arbitrary-value-oracles/manual-value/ManualValueOracleFactory.sol
* contracts/persistent/arbitrary-value-oracles/manual-value/ManualValueOracleLib.sol
* contracts/persistent/arbitrary-value-oracles/manual-value/ManualValueOracleProxy.sol
```

External Position: Arbitrary loans:

```
* contracts/persistent/external-positions/arbitrary-loan/ArbitraryLoanPositionLibBase1.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/ArbitraryLoanPositionDataDecoder.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/ArbitraryLoanPositionLib.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/ArbitraryLoanPositionParser.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/IArbitraryLoanPosition.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/modules/ArbitraryLoanFixedInterestModule.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/modules/ArbitraryLoanTotalNominalDeltaOracleModule.sol
* contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/modules/IArbitraryLoanAccountingModule.sol
* contracts/release/utils/MathHelpers.sol
```

External Position: Solv Finance v2 - convertible vouchers - buyer side:

```
* contracts/persistent/external-positions/solv-v2-convertible-buyer/SolvV2ConvertibleBuyerPositionLibBase1.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-buyer/ISolvV2ConvertibleBuyerPosition.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-buyer/SolvV2ConvertibleBuyerPositionDataDecoder.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-buyer/SolvV2ConvertibleBuyerPositionLib.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-buyer/SolvV2ConvertibleBuyerPositionParser.sol
* contracts/release/interfaces/ISolvV2ConvertibleMarket.sol
* contracts/release/interfaces/ISolvV2ConvertiblePool.sol
* contracts/release/interfaces/ISolvV2ConvertibleVoucher.sol
* contracts/release/interfaces/ISolvV2InitialConvertibleOfferingMarket.sol
```

External Position: Solv Finance v2 - convertible vouchers - issuer side:

```
* contracts/persistent/external-positions/solv-v2-convertible-issuer/SolvV2ConvertibleIssuerPositionLibBase1.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-issuer/ISolvV2ConvertibleIssuerPosition.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-issuer/SolvV2ConvertibleIssuerPositionDataDecoder.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-issuer/SolvV2ConvertibleIssuerPositionLib.sol
* contracts/release/extensions/external-position-manager/external-positions/solv-v2-convertible-issuer/SolvV2ConvertibleIssuerPositionParser.sol
* contracts/release/interfaces/ISolvV2ConvertiblePool.sol
* contracts/release/interfaces/ISolvV2ConvertibleVoucher.sol
* contracts/release/interfaces/ISolvV2InitialConvertibleOfferingMarket.sol
```

Update of DepositWrapper:

```
* contracts/release/peripheral/DepositWrapper.sol
```

Following files were part of a formatting change. It was only validated that no semantic changes to the files were made:

```
contracts/persistent/address-list-registry/AddressListRegistry.sol
contracts/persistent/dispatcher/IMigrationHookHandler.sol
contracts/persistent/external-positions/ExternalPositionFactory.sol
contracts/persistent/external-positions/ExternalPositionProxy.sol
contracts/persistent/global-config/GlobalConfigProxy.sol
contracts/persistent/global-config/bases/GlobalConfigLibBase1.sol
contracts/persistent/global-config/bases/GlobalConfigLibBaseCore.sol
contracts/persistent/global-config/utils/GlobalConfigProxyConstants.sol
contracts/persistent/protocol-fee-reserve/ProtocolFeeReserveProxy.sol
contracts/persistent/protocol-fee-reserve/bases/ProtocolFeeReserveLibBaseCore.sol
contracts/persistent/protocol-fee-reserve/utils/ProtocolFeeProxyConstants.sol
contracts/persistent/shares-splitter/SharesSplitterLib.sol
contracts/persistent/uint-list-registry/UintListRegistry.sol
contracts/persistent/vault/VaultProxy.sol
contracts/release/core/fund-deployer/FundDeployer.sol
contracts/release/core/fund/comptroller/ComptrollerLib.sol
contracts/release/extensions/external-position-manager/ExternalPositionManager.sol
contracts/release/extensions/external-position-manager/external-positions/aave-debt/AaveDebtPositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/aave-debt/IAaveDebtPosition.sol
contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/ArbitraryLoanPositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/arbitrary-loan/modules/ArbitraryLoanFixedInterestModule.sol
contracts/release/extensions/external-position-manager/external-positions/compound-debt/ICompoundDebtPosition.sol
```

```
contracts/release/extensions/external-position-manager/external-positions/convex-voting/IConvexVotingPosition.sol
contracts/release/extensions/external-position-manager/external-positions/liquity-debt/ILiquityDebtPosition.sol
contracts/release/extensions/external-position-manager/external-positions/the-graph-delegation/ITheGraphDelegationPosition.sol
contracts/release/extensions/external-position-manager/external-positions/uniswap-v3-liquidity/UniswapV3LiquidityPositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/uniswap-v3-liquidity/UniswapV3LiquidityPositionParser.sol
contracts/release/extensions/external-position-manager/external-positions/uniswap-v3-liquidity/interfaces/IUniswapV3LiquidityPosition.sol
contracts/release/extensions/fee-manager/FeeManager.sol
contracts/release/extensions/fee-manager/IFeeManager.sol
```

```
contracts/release/extensions/fee-manager/fees/ManagementFee.sol
contracts/release/extensions/fee-manager/fees/utils/FeeBase.sol
contracts/release/extensions/integration-manager/IIntegrationManager.sol
```

```
contracts/release/extensions/integration-manager/IntegrationManager.sol
contracts/release/extensions/integration-manager/integrations/utils/AdapterBase.sol
contracts/release/extensions/integration-manager/integrations/utils/IntegrationSelectors.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/AaveActionsMixin.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/CurveExchangeActionsMixin.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/CurveLiquidityActionsMixin.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/UniswapV3ActionsMixin.sol
contracts/release/extensions/integration-manager/integrations/utils/bases/CurveLiquidityAdapterBase.sol
contracts/release/extensions/policy-manager/PolicyManager.sol
contracts/release/extensions/policy-manager/policies/asset-managers/AllowedExternalPositionTypesPolicy.sol
contracts/release/extensions/policy-manager/policies/asset-managers/OnlyRemoveDustExternalPositionPolicy.sol
contracts/release/extensions/policy-manager/policies/utils/PolicyBase.sol
contracts/release/extensions/policy-manager/policies/utils/PricelessAssetBypassMixin.sol
contracts/release/infrastructure/gas-relayer/GasRelayRecipientMixin.sol
contracts/release/infrastructure/price-feeds/derivatives/feeds/UniswapV2PoolPriceFeed.sol
contracts/release/infrastructure/price-feeds/primitives/ChainlinkPriceFeedMixin.sol
contracts/release/infrastructure/staking-wrappers/StakingWrapperBase.sol
contracts/release/infrastructure/staking-wrappers/convex-curve-lp/ConvexCurveLpStakingWrapperFactory.sol
contracts/release/infrastructure/value-interpreter/ValueInterpreter.sol
contracts/release/utils/NonUpgradableProxy.sol
contracts/release/utils/beacon-proxy/BeaconProxy.sol
```

2.1.1 Excluded from scope

Only the files mentioned above are in scope. Goldfinch and FIDU are not in scope and are expected to work correctly as documented. Solv Protocol is not in scope and is expected to work correctly.

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 implements several new external positions to support lending uncollateralized loans to third parties, to integrate with Solv Finance v2 to buy, sell and claim convertible vouchers, and to integrate with Solv Finance v2's convertible vouchers as an issuer. Further, Avantgarde Finance implements a new price feed for FIDU from Goldfinch, two policies allowing setting fine-grained access control for managers in regard to their access on adapters and external positions, a manual value oracle that stores a value with a timestamp, a `uint256` list registry similar to the `address` list registry AddressListRegistry. Finally, Avantgarde Finance made an update to `DepositWrapper`.

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

2.2.1 UintListRegistry

The UintListRegistry, similar to the AddressListRegistry, is a system contract for storing lists of `uint256` values and identifying the by a `uint256` id. Anyone can create a list with any owner, any update type and any initial items. Hence, owners can attest the ownership by emitting events. Depending on the update type, the list owner may add and remove items from a list, transfer ownership and change the update type. Furthermore, the contract provides several helper functions for checking individual and multiple lists.

The following update types exist: `None`, `AddOnly`, `RemoveOnly`, `AddAndRemove`. The update type can be always changed to `None` and if the update type is `AddAndRemove` the update type can be changed to any one of them. `AddAndRemove` and `AddOnly` allow adding, `AddAndRemove` and `RemoveOnly` allow removing. The list owner can always be changed.

2.2.2 Policies

Two additional policies are introduced to the system to allow enforcing more fine-grained access control regarding fund managers:

1. Allowed adapters per manager: Maps a comptroller and manager address pair to a list of address list ids in the AddressListRegistry which specifies the adapters the manager can use for the fund of the comptroller. Called after any call on an integration.
2. Allowed external position types per manager: Maps a comptroller and manager address pair to a list of uint list ids in the UintListRegistry which specifies the external position types the manager can use for the fund of the comptroller. Called for any interaction with external positions (on creation, after a call on it, on reactivation and on removal).

Note that both policies can be disabled by the fund manager and that both `updateFundSettings` is implemented which allows the fund owner to update the policy through the PolicyManager.

2.2.3 Price Feed: FIDU

Goldfinch is an uncollateralized lending protocol where borrowers propose so-called borrower pools (however, they need to be pre-approved by so-called auditors). Lenders (called investors) can be either backers or liquidity providers. Backers invest with USDC in specific pools with first-loss capital while liquidity providers invest into a collective pool (senior pool) with second-loss capital whose allocation to the individual borrower pools is made according to the investments by the backers. When providing USDC, liquidity providers receive FIDU, a token representing a user's share for the pool. Backers may have a legal off-chain agreement with the borrowers, but it is not required.

Avantgarde Finance introduces a derivative price feed where the only supported asset is FIDU. The price is computed as the underlying amount of USDC minus the withdrawal fees that would occur on withdrawal of FIDU from the senior pool.

2.2.4 Nominated Owner Mixin

An abstract contract that implements a two-phase ownership transfer where the current owner nominates a future owner while the nominated owner needs to accept the ownership.

2.2.5 Manual Value Oracle

Manual value oracles are non-upgradeable proxy contracts deployed by the corresponding factory that store one updateable value along with the update time. More specifically, they offer the following functionality:

The manual value oracles

- `setUpdater()`: Sets the address that can update the oracle with a value. Callable by the oracle owner.
- `updateValue()`: Updates the oracle value and the timestamp.
- Getter functions to query the current value and the timestamp when it was set.

Note that it inherits from the nominated owner mixin and, hence, exposes its two-phase ownership transfer functionality. Further, anyone can deploy such oracles and anyone can be specified as an owner.

2.2.6 External Position: Arbitrary loans

Avantgarde Finance introduces a new type of external positions that allows creating uncollateralized loans for third parties. Each loan requires a separate external position and support only one borrower. While some accounting is done in the external position (keeping track of the borrowable amount, the totally borrowed amount and the totally repaid amount), the accounting logic for the interest model is moved to external accounting contracts which allows the arbitrary loan external position type to support different kinds of interest models.

Avantgarde Finance introduces the external position and offers two accounting modules. However, the fund manager can specify any address as the accounting module.

2.2.6.1 Arbitrary loans

The external position offers following functionality to fund managers:

- `ConfigureLoan`: Configures the loan. Allows to specify the borrower, the lent asset, the initial borrowable amount, the accounting module and its configuration data, and a description. Note that the accounting module could be specified as `0x0` which defines that no interest is expected to be repaid. If an accounting module is specified, its `configure()` function will be called. The description is solely emitted as an event for off-chain use.
- `UpdateBorrowableAmount`: Updates the borrowable amount to move funds from the vault proxy to the external position or vice-versa.
- `Reconcile`: Accounts any surplus balance of the lent asset as repaid and transfers the surplus to the vault proxy. Additional tokens can also be moved to the vault proxy. Before the repayment from the surplus is realized the `preReconcile()` hook is called on the accounting module (if one exists) to compute the amount that should be considered as repayment.
- `CloseLoan`: First, reconciles regularly and then updates the borrowable amount to 0 and sets the `isClosed` flag to true which blocks the increasing the borrowable amount (and closing the loan again). Note that after reconciling but before performing the closing actions an additional hook `preClose()` is called that allows specifying conditions when a loan can be closed.
- `CallOnAccountingModule`: Calls the `receiveCallFromLoan()` on the accounting module.

The external position offers the following functions to borrowers:

- `borrow()`: Only callable by the borrower to borrower the lent out asset (up to borrowable amount). Calls the `preBorrow()` hook on the accounting module (if one exists) and then updates the loan state on itself according to the borrowed amount. Funds are moved to the borrower.
- `repay()`: First the `preRepay` hook on the accounting module is called (if one exists and if not the maximum `uint256` is treated as full repayment) to compute the repaid amount. Second, the repaid amount is increased and funds are transferred from the `msg.sender` to the vault proxy. Note that any loan can be repaid by anyone.

Note that if an accounting module is defined it will determine the actual amount that is being repaid (through reconciliation or repayment). That allows treating full repayments with `uint256.max` in the accounting module (similar to the default case when no module is defined).

Note that the external position implements the following two methods to comply with the interface:

- `getDebtAssets()`: Since no debt is created, this will return two empty arrays.
- `getManagedAssets()`: If there is an accounting module it calls its `calcFaceValue()` function and returns its result. Otherwise, the outstanding loan balance is returned. Note that direct transfers to the contract will be only considered to a certain degree.

2.2.6.2 Accounting modules

Accounting modules offer the following functionality:

- `calcFaceValue()`: Calculates the face value of the loan.
- `configure()`: Function to configure the loan.
- `preBorrow()`: Hook called before the effect of borrowing to perform any accounting necessary for the interest model.
- `preRepay()`: Hook called before the effect of repayment to perform any accounting necessary for the interest model and return how much is repaid (e.g. handling `uint256.max`).
- `preReconcile()`: Hook called before the effect of reconciliation to perform any accounting necessary for the interest model and return how much is repaid (e.g. handling `uint256.max`).
- `preClose()`: Hook called before the effect of closing to enforce conditions that the loan can be closed.
- `receiveCallFromLoan()`: Function to support calls from the loan contract.

Avantgarde Finance implements the following accounting modules:

- *Total Nominal Delta Oracle module*: Allows an arbitrary oracle (that implements the interface of the manual value oracle and is specified on configuration) to return the interest owed. If a staleness threshold is defined, it is checked against the recency of the oracle value.
- *Fixed Interest module*: The owed amount is computed with a fixed interest rate per second from the owed amount. Note that one regular interest rate and one for late repayments can be specified and are handled accordingly. Both are denominated in the RAY base.

Both accounting modules support multiple loans.

2.2.7 Uint256ArrayLib

Uint256ArrayLib is an internal library for providing operations on arrays containing `uint256` items similar to what AddressArrayLib implements for arrays containing `address` items. First, it is used in the following external positions for Solv.

2.2.8 External Positions: Solv Finance v2 - convertible vouchers

Solv Protocol is a protocol that introduces so-called convertible vouchers. These combine a zero-coupon bond and price range such that fixed yield can be offered. Issuers can define their own payout curve to specify how payouts according to the settlement price will be handled. Issuers lock tokens for the voucher and can offer them on initial convertible voucher offering market (convertible IVO market) while buyers can buy them. Additionally, there is a secondary market, the convertible voucher marketplace, where buyers can buy and sell the vouchers. Note, that vouchers can be claimed after maturity and either the locked token or a stable coin will be received by the holder depending on the payout curve. Convertible vouchers implement EIP-3525.

Please consider the relevant parts of Solv V2's documentation for further details.

Avantgarde Finance implements two new external positions to integrate with Solv's convertible vouchers. One to allow interacting with Solv V2's system from the non-issuer buyer side, and one to allow interacting with from the issuer side.

2.2.8.1 Buyer side

The buyer side external position offers the following actions:

- `BuyOffering`: Buys an offering from the convertible IVO market using the currency specified in the offering. The external position receives a token from the voucher representing its ownership of the bought amount of units.

- `BuySaleByAmount`: Buys a sale from the convertible voucher marketplace with a given amount of the currency specified in the sale using function `buyByAmount()` on the marketplace contract. The external position receives a token from the voucher representing its ownership of the bought amount of units.
- `BuySaleByUnits`: Buys an amount of units from the sale in the convertible voucher marketplace with the currency specified in the sale using function `buyByUnits()` on the marketplace contract. The external position receives a token from the voucher representing its ownership of the bought amount of units. Note that if all units are bought the token id set in the sale is transferred to the external position. Otherwise, a new token id is generated for the units bought.
- `Claim`: After maturity, vouchers can be claimed by calling `claimTo()` on the voucher contract. This action claims the voucher and sends the funds received to the vault proxy. The fund manager can specify how many units to claim. Note that either the underlying token or the fund currency can be received.
- `CreateSaleDecliningPrice`: Creates a sale with a declining price on the secondary marketplace by calling `publishDecliningPrice()`. Moves the voucher to the marketplace contract. Note that when third parties buy vouchers, the funds are sent to the external position.
- `CreateSaleFixedPrice`: Creates a sale with a fixed price on the secondary marketplace by calling `publishFixedPrice()`. Moves the voucher to the marketplace contract. Note that when third parties buy vouchers, the funds are sent to the external position.
- `Reconcile`: When units are bought from the sale, the external position will receive tokens. This action collects all tokens that could have been received through sales and sends them to the vault proxy.
- `RemoveSale`: Removes the sale from the marketplace by calling `remove()`. Note, that this action also reconciles the token that could have been received through sales and sends it to the vault proxy.

Note that the external position implements the following two methods to comply with the interface:

- `getDebtAssets()`: Since no debt is created, this will return two empty arrays.
- `getManagedAssets()`: Accumulates the claimable amounts from all vouchers held or sold and considers the reconcilable balances. Reverts if one voucher has not reached maturity.

2.2.8.2 Issuer side

The issuer side external position offers the following actions:

- `CreateOffer`: Creates an offering on the convertible IVO market by calling `offer()` on it. Moves the voucher's underlying. If units are sold, the specified currency will be received directly by the external position.
- `Reconcile`: Since funds from sales will directly be received by the external position, this action allows to collect all tokens received to be sent to the vault proxy.
- `Refund`: Refund allows to fund the voucher with the so-called `fundCurrency`. This will let holders receive the `fundCurrency` instead of the underlying if the right conditions are met. Call `refund()` on the voucher's convertible pool and sends the fund currency there.
- `RemoveOffer`: Removes an offer from the IVO market by calling remove `remove()` on it. Could receive some underlying tokens from unsold units which it forwards to the vault proxy. Additionally, reconciles the sale currency and sends it to the vault proxy.
- `Withdraw`: Withdraws the amounts of the fund currency and the underlying not needed for buyers payouts considering the payout curve to the external position with a call to the convertible pool's `withdraw()` function. Sends the received funds to the vault proxy.

Note that the external position implements the following two methods to comply with the interface:

- `getDebtAssets()`: Since no debt is created, this will return two empty arrays.
- `getManagedAssets()`: Accumulates the withdrawable (claimable through `Withdraw`) and the refundable (claimable through `remove()`) amounts from all vouchers issued and considers the reconcilable balances. Reverts if one voucher has not reached maturity.

2.2.9 Deposit Wrapper

The `DepositWrapper` contract allows exchanging ETH for a fund's denomination asset and buying shares of the fund.

2.2.10 Trust Model

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

The Fund Manager is fully trusted given the reasons detailed in the previous reports. However, given the arbitrary loans, the fund manager is even more trusted. Note that Solv's convertible vouchers can be redeemed only after maturity has been reached and, hence, the fund manager, as mentioned in previous audits, is fully trusted to keep the funds balanced in such way that users can withdraw their shares.

All external systems are expected to be non-malicious and work correctly as documented. For Solv `Market.asset == Voucher.underlying()` is expected to hold. Fund managers are expected to not only behave honestly but also to fully understand the systems they are interacting with which includes choosing appropriate parameters.

Any party and any address interacting with arbitrary loans is fully trusted which also includes the borrower.

In general we assume Enzyme only interacts with normal ERC-20 tokens without any special behavior including rebasing, multiple entry points or callbacks.

2.2.11 Changes in version 3

- It was specified that Solv external positions are expected to revert if maturity has not been reached.
- Managers that do not have any configured lists could use any adapter or external position. This changed such that a manager without any configured list cannot use any adapter or external position. Moreover, a manager can still be authorized to use any adapter or external position type if the first entry in the array of list IDs is a bypass flag.

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	0
Low-Severity Findings	1

5.1 Solv's BUYER_PAY Fee Pay Type Is Unsupported

Correctness Low Version 1 Code Partially Corrected Risk Accepted

When buying vouchers from the marketplace, fees are paid. Note that Solv has two fee pay types such that either the buyer or the seller pays fees. If the buyer pays, the fee is added to the amount transferred from the buyer. Note that Solv's internal function `_buy()` will transfer `transferInAmount` from the buyer which is defined as `amount_.add(fee_)`.

The external position for the buyer side does not consider that which leads to the following consequences:

1. Action `BuySaleByAmount` is not supported if the fee type is BUYER_PAY as the approval made will be insufficient.
2. Action `BuySaleByUnits` is not supported if the fee type is BUYER_PAY as the funds sent from the vault will not be sufficient to perform the action.

with the exception that if some unreconcilled funds are available to the external position, the funds could be sufficient to perform the action.

Code partially corrected:

1. Not corrected: Note that buying by amount on Solv will not transfer in the passed in amount but the passed in amount plus fees. However, `BuyByAmount` does not consider fees. See the code of SolvConvertibleMarket.sol file here https://etherscan.io/address/0x29935f54a45f5955ad7bc9d5416f746c3d1b9d69 on line 502.

```
if (vars.feePayType == FeePayType.BUYER_PAY) {
  vars.transferInAmount = amount_.add(fee_);
  ...}
...
```

```
ERC20TransferHelper.doTransferIn(
      sale_.currency,
      buyer_,
      vars.transferInAmount
  );
```

Ultimately, insufficient funds could be moved and the approval given to Solv could be insufficient.

2. Code corrected: The code has been adapted such that the fee is in included in the transferred in amount.

Note that the fee computation made for the `BuyByUnits` action could be off. There is a special case where the voucher's underlying could be also the currency. In such situations the fee is computed differently and is based on `repoFeeRate` instead of the market's `feeRate`.

Risk accepted:

Avantgarde Finance states the following:

```
the Solv team says that they will upgrade to the version of `SolvConvertibleMarket` that is in their GitHub repo (b207d5e), which fixes this issue
(buyer fee is deducted from `amount`, and there is no longer a `repoFeeRate`). The Enzyme Council will assure that the upgrade has occurred before
adding the external position type. Even if no upgrade were to occur, the worst case is that `BuySaleByAmount` will revert when there is a buyer fee,
which does not result in value loss for the fund.
```

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

- Solv Issuer Double Accounting Code Corrected

Medium-Severity Findings	5

- Full Balance Is Pushed on Reconciliation Code Corrected
- Offer ID and Voucher Mismatch Code Corrected
- Solv Finance: No Support for Raw ETH as Currency Code Corrected
- Solv Issuer Ignores Possibly Withdrawable Voucher Slots Code Corrected
- getManagedAssets for Solv Buyer Side Reverts if Maturity Not Reached Specification Changed

Low-Severity Findings	2

- Incomplete NatSpec for ManualValueOracleLib.init() Code Corrected
- assetsToReceive Not Containing Assets Code Corrected

6.1 Solv Issuer Double Accounting

Correctness High Version 1 Code Corrected

The Solv Issuer external position keeps track of the offered vouchers on the convertible offering marketplace. To compute their values it sums up

1. The token amounts that could be withdrawn by the issuer with internal function `__getWithdrawableAssetAmounts`.
2. The token amounts that could be claimed in case some units are still held with internal function `__getOffersUnderlyingBalance()`
3. The unreconciled token amounts.

`__getWithdrawableAssetAmounts()` iterates over all offers and further iterates over all issuer slots of the external position. There is a possibility of accounting withdrawable amounts multiple times.

```
for (uint256 i; i < offersLength; i++) {
    // ...
    ISolvV2ConvertibleVoucher voucherContract = ISolvV2ConvertibleVoucher(
        INITIAL_CONVERTIBLE_OFFERING_MARKET_CONTRACT.offerings(_offers[i].offerId).voucher
    );
    ISolvV2ConvertiblePool voucherPoolContract = ISolvV2ConvertiblePool(
        voucherContract.convertiblePool()
    );
    uint256[] memory slots = voucherPoolContract.getIssuerSlots(address(this));
    // ...
```

```
    for (uint256 j; j < slotsLength; j++) {
        (uint256 withdrawCurrencyAmount, uint256 withdrawTokenAmount) = voucherPoolContract
            .getWithdrawableAmount(slots[j]);
        // logic for summing up
        // ...
```

Consider the following example:

1. First offer is created with the voucher being X such that it has slot id 1.
2. Second offer is created with the voucher being X such that it has slot id 2.
3. The above code is executed.
4. The convertible pool of the voucher gives the slots 1 and 2.
5. The withdrawable amount is added twice since the inner loop for both offers will iterate over slot ids 1 and 2 and add the withdrawable amounts twice.

Ultimately, the withdrawable amounts may be added multiple times in the evaluation.

Code corrected:

Now, not only offers are tracked but also issued voucher addresses. Hence, estimating the withdrawable amounts is done by iterating now over the issued voucher addresses which do not contain duplicates.

6.2 Full Balance Is Pushed on Reconciliation

Design | Medium | Version 1 | Code Corrected

To handle direct token transfers to the loan contract as repayments (and to handle arbitrary tokens received), Avantgarde Finance introduces a reconciliation functionality for arbitrary loans, which allows moving arbitrary tokens received (e.g., insurance payments) to be moved to the vault. Furthermore, it considers all surplus balance (compared to the borrowable amount) of the loan token as a repayment. However, it always moves the full loan token balance to the vault. While this makes sense when closing the vault, it may break the loan's logic when action reconcile is executed (e.g., borrowable amount > 0 but borrows are impossible).

Code corrected:

Reconciliation for the `Reconcile` action and reconciliation for the `Close` action are now performed differently. A boolean `_close` argument was added to the `__reconcile` function to make this distinction.

6.3 Offer ID and Voucher Mismatch

Design | Medium | Version 1 | Code Corrected

When buying an offer from the Solv IVO, the fund manager can specify the voucher address and the offering ID. However, the voucher address could mismatch with the offer's voucher stored in the `Offering` struct.

Consider the following scenario:

1. Fund manager inputs an offer id such that `Offer.voucher` and the input `voucher` mismatch.

2. The SolvV2ConvertibleBuyerPositionParser specifies `Offer.currency` as the asset to transfer while specifying the amount to transfer as

```
uint256 amount = uint256(units).mul(voucherPrice).div(10**uint256(market.decimals));
```

3. The `nextTokenId()` is queried on the wrong voucher and the contract maximum approval is given to the IVO market.
4. `buy()` is called on the IVO market. As long as the amount computed in step 2. is sufficient, buying will succeed.
5. The approval is revoked.
6. The input `voucher` and the token id from step 3 are pushed on the position's `offers` array.

While it requires an error by the fund manager, it could have consequences such as

- tracking of a wrong voucher and token id leading to wrong estimations of the total value,
- stuck tokens due to high amounts being moved also leading to wrong fund evaluations,
- being stuck with the wrong voucher and token id
- potential of double tracking of voucher and token id

Ultimately, to buy an offering it could be sufficient to specify solely the units and the offering id.

Code corrected:

The voucher address is not an action argument anymore for buying from IVOs but is retrieved from the offering. Hence, the position parser and the position logic have been adapted accordingly.

6.4 Solv Finance: No Support for Raw ETH as Currency

Design | Medium | Version 1 | Code Corrected

Raw ETH is not supported as a currency for the Solv convertible vouchers. This is problematic because Solv supports ETH through the `doTransferOut` function in the `ERC20TransferHelper` library, which uses a special constant address `ETH_ADDRESS` for such raw currency transfers. Given the lack of sanity checks for the assets in a voucher, it could be possible that such a voucher becomes unredeemable (e.g. claim action while fund currency is ETH).

Code corrected:

The function `__validateNotNativeToken` was added to verify that the asset's address is not equal to the special value `NATIVE_TOKEN_ADDRESS`.

6.5 Solv Issuer Ignores Possibly Withdrawable Voucher Slots

Correctness | Medium | Version 1 | Code Corrected

When creating an IVO offer, a slot is created for the issuer and the offer which gives it a uniqueness property given the slot details. Action `Withdraw` allows the fund manager to claim assets from the voucher contract after maturity while action `Remove` removes the offer from the offering market such that the overhead underlying for the unsold units is refunded. Moreover, `Remove` removes the offer from the `offers` array such that it becomes untracked. While it is still possible to call `Withdraw`, the value of `getManagedAssets` has dropped even though assets could still be withdrawn.

Consider the following scenario:

1. An offer is created. Some units were sold but not all.
2. The offer can be removed and there are withdrawable amounts. Assume the refund amount is 10 X and the withdrawable amounts are 10 X and 10 Y.
3. `getManagedAssets()` return 20 X and 10 Y.
4. `Remove` is executed. 10 X are moved to the vault proxy.
5. `getManagedAssets()` returns 0.
6. `Withdraw` is executed. The fund's value rises suddenly.

In general, such behaviour could be introduced.

Code corrected:

Now, not only offers are tracked but also issued voucher addresses. Removing an offer does not remove the issued voucher and, thus, the voucher's slots remain tracked.

6.6 `getManagedAssets` for Solv Buyer Side Reverts if Maturity Not Reached

Design | Medium | Version 1 | Specification Changed

`getManagedAssets` evaluates the value held by the position. To do so, it iterates over all vouchers held, currently held or being sold, and computes their value with internal method `__getClaimableAmount()` which contains the following code:

```
uint128 settlePrice = poolContract.getSettlePrice(slotId);
require(settlePrice > 0, "Price not settled");
```

Note that Solv's convertible pool contract implements `getSettlePrice()` such that it reverts if maturity has not been reached or if the price is negative. Ultimately, no voucher that has not reached its maturity can be evaluated and hence `getManagedAssets()` will revert which will block several operations in the Enzyme system. Even if `getSettlePrice()` did not fail, the second requirement may lead to reverts.

Specification changed:

Avantgarde Finance states:

```
This is an architectural decision to revert upon price lookup for all Solv vouchers (in both
Buyer and Issuer features) that are issued or held prior to maturity, rather than estimate the
value of an unsettled voucher. Price-dependent fund functions will revert while any such voucher
is issued/held.
```

6.7 Incomplete NatSpec for `ManualValueOracleLib.init()`

Design Low Version 1 Code Corrected

The documentation of `ManualValueOracleLib.init()` is:

```
/// @notice Initializes the proxy
/// @param _owner The owner of the oracle
/// @param _updater The updater of the oracle value
```

Note that the `_description` parameter is not documented and, thus, the NatSpec is incomplete.

Code corrected:

The description parameter has been added to the NatSpec.

6.8 `assetsToReceive` Not Containing Assets

Correctness Low Version 1 Code Corrected

Both the parsers for Solv V2 Buyer positions and Solv V2 Issuer positions could lead to untracked assets. The library used for adding items to memory arrays, creates a new memory array with the old and new items. However, in some occasions, the return value is not assigned to `assetsToReceive` after an item is added. Hence, it could be possible the assets remain untracked.

Note that Avantgarde Finance reported the issue.

Code corrected:

The return values are assigned.

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 Arbitrary Loan Powers

Note Version 1

While all addresses involved in the arbitrary loan mechanism are fully trusted, such external positions may give managers very high control about the fund. Some (incomplete) list examples:

1. Stealing funds very easily by giving the full balance as a loan to itself.
2. Manipulating the valuation of the fund to profit by specifying an accounting module that computes the face value when queried as extremely high.
3. Increase the number of shares by using the loan to invest in the fund.
4. Reentrancy possibilities.
5. Blocking behaviour.