PUBLIC

Code Assessment

of the Sulu Extensions IX Smart Contracts

February 21, 2023

Produced for

by

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 9
4 Terminology 10
5 Findings 11
6 Resolved Findings 13
7 Informational 14
8 Notes 15

1 Executive Summary

Dear Mona and Sean,

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

Avantgarde Finance implemented changes to the existing UniswapV2 (support for tokens with fees on transfer) and Balancer (`batchSwap()`) adapters. New external positions for Solv bonds have been added, similar to the existing external position for Solv convertibles but without support for the secondary market. A new gated shares wrapper with a redemption queue has been added.

The most critical subjects covered in our audit are functional correctness, access control, and integration with external protocols.

The general subjects covered are code complexity, upgradeability, and documentation. 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	0
Medium-Severity Findings	1
• Risk Accepted	1
Low-Severity Findings	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 IX 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	30 January 2023	44b40cb79df5efe9aa302f440a487d1282cf5854	Initial Version
2	5 February 2023	4101e09fcc60f77a54071f5056b02f17b5d3ff86	Gated shares wrapper
3	14 Feburary 2023	f04f598f9b88891d1aa0694c01b715e34421397f	Final Version

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

The following files and directories were in scope

Adapter Update: Make UniswapV2 swaps work with fee-on-transfer tokens

```
contracts/release/extensions/integration-manager/integrations/adapters/UniswapV2ExchangeAdapter.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/UniswapV2ActionsMixin.sol
contracts/release/interfaces/IUniswapV2Router2.sol
```

Adapter Update: Add `batchSwap()` support to BalancerV2LiquidityAdapter

```
contracts/release/extensions/integration-manager/integrations/adapters/AuraBalancerV2LpStakingAdapter.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/BalancerV2ActionsMixin.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/StakingWrapperActionsMixin.sol
contracts/release/extensions/integration-manager/integrations/utils/bases/BalancerV2LiquidityAdapterBase.sol
contracts/release/interfaces/IBalancerV2Vault.sol
```

External Position: Solv v2 bonds

```
contracts/persistent/external-positions/solv-v2-bond-buyer/SolvV2BondBuyerPositionLibBase1.sol
contracts/persistent/external-positions/solv-v2-bond-issuer/SolvV2BondIssuerPositionLibBase1.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-buyer/ISolvV2BondBuyerPosition.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-buyer/SolvV2BondBuyerPositionDataDecoder.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-buyer/SolvV2BondBuyerPositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-buyer/SolvV2BondBuyerPositionParser.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-issuer/ISolvV2BondIssuerPosition.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-issuer/SolvV2BondIssuerPositionDataDecoder.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-issuer/SolvV2BondIssuerPositionLib.sol
contracts/release/extensions/external-position-manager/external-positions/solv-v2-bond-issuer/SolvV2BondIssuerPositionParser.sol
contracts/release/interfaces/ISolvV2BondPool.sol
contracts/release/interfaces/ISolvV2BondVoucher.sol
```

Peripheral: Gated shares wrapper

```
contracts/persistent/global-config/GlobalConfigLib.sol
contracts/persistent/global-config/interfaces/IGlobalConfig2.sol
contracts/persistent/global-config/interfaces/IGlobalConfigLibComptrollerV4.sol
contracts/persistent/shares-wrappers/gated-redemption-queue/GatedRedemptionQueueSharesWrapperFactory.sol
contracts/persistent/shares-wrappers/gated-redemption-queue/GatedRedemptionQueueSharesWrapperLib.sol
contracts/persistent/shares-wrappers/gated-redemption-queue/bases/GatedRedemptionQueueSharesWrapperLibBase1.sol
contracts/persistent/vault/interfaces/IVaultCore.sol
contracts/release/core/fund/vault/IVault.sol
contracts/release/core/fund/vault/VaultLib.sol
```

2.1.1 Excluded from scope

All files not mentioned above are not in scope. UniswapV2, Balancer and Solv are not in scope and are expected to work correctly as documented.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Avantgarde Finance implements integrations with various external systems. Some integrations were newly added, others were updated.

2.2.1 UniswapV2 Adapter: Support of swaps with fee-on-transfer tokens

The adapter now calls `swapExactTokensForTokensSupportingFeeOnTransferTokens()` instead of `swapExactTokensForTokens()` on the UniswapV2 Router. This allows the adapter to support tokens with fees on transfer such as the PAXG token.

2.2.2 BalancerV2LiquidityAdapterBase: Add `batchSwap()`

`takeOrder()` which uses Balancer's batchSwap function has been added to BalancerV2LiquidityAdapterBase. This base contract is inherited by two Adapter contracts, AuraBalancerV2LpStakingAdapter and BalancerV2LiquidityAdapter. Being able to execute batchwaps is needed to support LPing to nested Composable Stable Pools like Balancer Boosted Aave USD Pool (bb-a-USD).

Optionally allows combining "unstake BPT + batchSwap()" and "batchSwap() + stake BPT", i.e., mimicking the functionality of lendAndStake() and unstakeAndRedeem().

2.2.3 External Position: Solv v2 bonds

Solv Protocol describes itself as a Web3 liquidity infrastructure that utilizes Solv Payable, a full-suite semi-fungible token solution to enable institutional entities and retail users to access liquidity by creating, issuing, or trading semi-fungible tokens in a zero-trust and transparent way.

Issuers create offers for bond vouchers on the initial voucher offering market where buyers can buy them.

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 initial voucher offering market for bond vouchers. A similar external position for Solv v2 convertibles already exists.

Note that contrary to the external position for convertibles, this external position does not support trades on the secondary market.

2.2.3.1 Bond Issuer:

- `CreateOffer`: Creates an offering on the IVO market by calling `offer()` on it. An offer for a fund currency must be collateralized using the underlying (currency of the voucher) transferred from the vault. This must match the `market.asset` of the voucher stored in the IVO market contract. The current implementation of the external position can't provide another asset. Units to be sold depend on the amount of underlying supplied and the lowest price specified. When units are sold, the tokens 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. This will let holders receive the fund currency. Calls `refund()` on the voucher's bond pool and sends the fund currency there.
- `RemoveOffer`: Removes an offer from the IVO market by calling `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. This function should also be called if an offer successfully sold all units in order to delete the offer stored in the external position.
- `Withdraw`: Withdraws the underlying not needed for buyers' payouts to the external position with a call to the bond 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. **Caution:** Intentionally reverts if one voucher has not reached maturity.

2.2.3.2 Bond Buyer:

- `BuyOffering`: Buys the specified amount of the specified offerId from the IVO market. Offers requiring Ether are not supported. The external position receives a token from the voucher representing its ownership of the bought amount of units.
- `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, the fund currency or in the corner case of `isIssuerRefunded=true` and `settlePrice > slotDetail.highestPrice`, both can be received.

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. **Caution:** Intentionally reverts if one voucher has not reached maturity.

2.2.4 Peripheral: Gated shares wrapper

Shares wrapper for Enzyme v4 and future versions. Handles deposits and redemption according to certain rules:

- The manager of the wrapper may force redemptions
- The manager can optionally require that deposits, redemptions or transfers require prior approval from the manager. Deposits and Transfers must use exactly the approval amount specified. Removals may use at most the approval amount, but can also use less. Each approval can only be used once and is removed afterward, even if not using the full amount. The manager can also give an infinite approval, which allows any size and any amount of the specified action for an address.
- There is a `relativeSharesCap` that limits how much of the total supply of shares may be withdrawn in each epoch. Consider an example where the relativeSharesCap is 25%, the total supply of shares is 1000, and the total shares requested by all users is 300. This means the `absoluteCap` of shares for that epoch will be `25%*1000=250` shares. When the admin executes the redemptions, each user that is processed receives `250/300=83.3%` of the shares that they currently have requested. A user that has requested 100 shares, will receive 83.3 shares. A user that has requested 200 shares will receive 166.6 shares.

- The manager defines a redemption window frequency and duration. Users can only make requests outside of the redemption window duration. Requests can be serviced once per user in each redemption window.

The shares wrapper has the following main functions:

- `deposit` allows a user to deposit to an enzyme vault in return for ERC-20 shares wrapper tokens. It must be a single-asset deposit. For Enzyme V4 vaults it is enforced that this must be the denomination asset.
- `requestRedeem` allows a user to request that their shares be redeemed in a single-asset redemption from the vault when the next redemption window happens. It may take multiple windows until the request is completely fulfilled. This function must be called outside of a redemption window.
- `cancelRedeem` allows a user to cancel their redemption request.
- `redeemFromQueue` allows the manager to execute requested redemptions. Depending on the inputs, either all users in the queue or only some of them will be processed. This means that if the queue is long, redemptions can be split over multiple calls. Each processed user will receive their pro-rata share of requested shares, out of the total available shares allowed for redemption. If the manager does not call `redeemFromQueue` in a redemption window, users will not be able to withdraw. If the manager calls it in a later window, users will not receive extra redemptions due to having been skipped earlier.
- `kick` allows the manager to immediately force a user to redeem their wrapper tokens. This forces an immediate full redemption, ignoring any redemption limits.

2.2.5 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 it is now accepted that unsupported assets may be added as tracked asset to a vault. E.g. this may happen when an external position returns assets to the vault. Since no price feed for such assets exists in the system, calculating the GAV or share price will fail. This can be resolved by untracking or trading away the unspported asset. Ultimately it's the fund owner's responsabilty to handle this appropriately.

In general, we assume Enzyme only interacts with normal ERC-20 tokens that do not have multiple entry points, callbacks or other special behaviours.

Fund owners and asset managers are generally fully trusted for a fund. However, their powers can be limited through the fund's settings. The funds settings/policies are assumed to be set up correctly for the intended configuration / usage.

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.

All external systems are expected to be non-malicious and work correctly as documented. Note that Solv is fully upgradeable and hence may change its behavior. Notably, for the Solv Bond Issuer External Position in the InitialConvertibleOfferingMarket we assume `market.asset` of the voucher is set to the `underlying` of the voucher, the external position doesn't support to provide another asset as collateral. INITIAL_BOND_OFFERING_MARKET_CONTRACT must be correctly initialized, the vouchers are assumed to be Bond Vouchers.

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	1

- Preferential Withdrawal Risk Accepted

Low-Severity Findings	1

- BondBuyer: Claims Involving Ether Track Wrong Asset Acknowledged

5.1 Preferential Withdrawal

Design Medium Version 1 Risk Accepted

When there are more withdrawal requests than can be serviced, all users receive the same percentage of their withdrawals. A user that wants to make a partial withdrawal could take advantage of this.

Consider an example where withdrawal requests are fulfilled at 50%. A user that wants to withdraw 100 shares could instead request to withdraw 200 shares (given he has enough shares). Their request would be fulfilled by half, giving them 100 shares. Now they can cancel the remaining withdrawal request.

In this way, the user was able to circumvent the withdrawal limit at no cost. Other users were able to withdraw fewer shares than they would have otherwise.

Risk accepted:

Avantgarde Finance states:

```
This is the intended behavior. Also note that redeemers who request to redeem more
than they actually would like to redeem are risking that their entire requested
amount be redeemed in full if the cap is not met, the cap is updated by the manager,
or other redeemers cancel their requests.
```

5.2 BondBuyer: Claims Involving Ether Track Wrong Asset

Correctness Low Version 1 Acknowledged

Claiming a position involving Ether will result in the wrong asset being added to the tracked assets of the vault. Note that the vault actually supports receiving Ether (it immediately wraps it as WETH).

Consider the parser of the SolvV2BondBuyerPosition:

```
else if (_actionId == uint256(ISolvV2BondBuyerPosition.Actions.Claim)) {
        (address voucher, uint256 tokenId, ) = __decodeClaimActionArgs(_encodedActionArgs);

        ISolvV2BondVoucher voucherContract = ISolvV2BondVoucher(voucher);

        uint256 slotId = voucherContract.voucherSlotMapping(tokenId);
        ISolvV2BondPool.SlotDetail memory slotDetail = voucherContract.getSlotDetail(slotId);

        assetsToReceive_ = new address[](2);
        assetsToReceive_[0] = voucherContract.underlying();
        assetsToReceive_[1] = slotDetail.fundCurrency;
```

For arbitrary vouchers, one of the assets may be Ether as Ether is technically supported by the Solv v2 smart contracts.

Solv v2 represents the Ether asset as "0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE" which in this case will be added to `assetsToReceive`. Within Enzyme, however, the correct asset to track in this case would be the address of WETH.

This results in an unsupported asset being tracked by a vault which may have severe consequences. For example, it breaks `calcGav()`.

Whether such vouchers actually exist depends on the market configurations administrated by Solv Protocol. These markets may change in the future. Since the external position may interact with any offer on the IVOMarket / any voucher, such an issue may arise.

The InitialVoucherOfferingMarket currently doesn't support to create offers with Ether as underlying since `offer()` misses the `payable` modifier. Note that the implementation otherwise supports the case to handle `Ether`.

The currency of a voucher may be Ether. Contrary to `offer()` `buy()` features the `payable` modifier and hence such vouchers can be bought successfully. Note that one can't buy such a position in Ether via the external position since it doesn't support providing ERC20 tokens. This however doesn't prevent all scenarios where `claim()` may return Ether as such an NFT may be transferred directly to the external position.

Acknowledged:

Avantgarde Finance states:

```
While ETH can technically be the currency of the offer, Solv’s refund
logic depends on it being a stablecoin:
https://github.com/solv-finance/solv-v2-ivo/blob/
ac12b7f91a7af67993a0501dc705687801eb3673/vouchers/bond-voucher/
contracts/BondPool.sol#L174
```

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	0

6.1 Missing `indexed` in Event

Informational Version 1 Code Corrected

The event `initialized` emitted in `GatedRedemptionQueueSharesWrapperLib.init()` contains the address of the VaultProxy. This field is not indexed, hence one can't easily search such events for a certain VaultProxy. Given that the Factory doesn`t implement access control when deploying new shares wrappers it may be helpful to have this field indexed so that one can more easily search the events.

Code corrected:

The parameter of the event has been indexed.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Number of Assets

Informational **Version 1**

By design, the external position framework adds all assets specified as incoming assets to the tracked assets of the vault, regardless whether the vault has a non-zero balance at the end of the operation.

Notably `ISolvV2BondBuyerPosition.Actions.Claim` adds two assets as both may be received. In a corner case scenario, despite actually receiving one asset only, adding two may exceed the position limit and hence the operation fails.

7.2 OpenZeppelin ERC20 Hooks

Informational **Version 1**

The GatedRedemptionQueueSharesWrapperLib overrides `transfer()`/`transferFrom()` in order to validate the transfer (`__preProcessTransfer`).

The OpenZeppelin ERC20 implementation provides a hook, (`_beforeTokenTransfer`) which could be used for this. Note that this hook is also executed upon minting/burning. For more information please refer to documentation of OpenZeppelin:

- https://docs.openzeppelin.com/contracts/3.x/extending-contracts#using-hooks

8 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.

8.1 Bond Buyer Requires Trusted Fund Manager

Note Version 1

Investors need to trust the fund manager to a certain degree.

A fund manager can always drain funds e.g. through bad trades. This is documented: https://specs.enzyme.finance/topics/known-risks-and-mitigations#opportunistic-managers

Note that this is amplified when a fund can use the SolvV2BondBuyer External Position: A malicious fund manager may create an IVO offer via the Solv Protocol with a very high `lowestPrice` set for their collateral asset. Then they can buy this offer through the External Position and never pay back the principal to the Bond. This would leave the fund with a small amount of collateral, while the fund manager could keep all value that was in the fund.

8.2 Deployment of GatedRedemptionQueueSharesWrapper

Note Version 1

Anyone may deploy a GatedRedemptionQueueSharesWrapper for any fund through the factory. This includes setting the initial configuration. For example, a deployer can set themselves as manager.

Users and fund owner should be aware and excercise extra caution. The owner of a fund has full control over any such GatedRedemptionQueueSharesWrapper and can reconfigure it.

Multiple SharesWrapper can be deployed for the same fund. Note that they all bear the same `name` and `symbol`.

8.3 Kick Ignores Redemption Limit

Note Version 1

The `kick` function in the shares wrapper allows an admin to immediately force a user redemption. This ignores the redemption limit.

Note that the limit of other users' withdrawals is not reduced by this, so the maximum redeemed amount in that period can be the redemption limit, plus any `kick` actions in addition.

8.4 Redemption Requests Not Always Possible

Note Version 1

Note that redemption requests to the shares wrapper can only be made outside of the redemption window.

The comments in the code suggest that window frequency could be chosen every 2 weeks and duration 1 week:

```
    struct RedemptionWindowConfig {
    uint64 firstWindowStart; // e.g., Jan 1, 2022; as timestamp
    uint32 frequency; // e.g., every 2 weeks; in seconds
    uint32 duration; // e.g., 1 week long; in seconds
    uint64 relativeSharesCap; // 100% is 1e18; e.g., 50% is 0.5e18
}
```

With these settings, users would only be able to make redemption requests half of the time. If a user is unlucky, they would need to wait for an entire week until they can make a transaction that does not revert.

8.5 Vault May Track Unsupported Assets

Note **Version 1**

The external position framework relies on the parser of the external position to check the assets returned as `_assetsToReceive`. The code of the external position framework doesn't do any checks itself and simply adds any asset to the tracked assets of the vault. Note that this is also independent of the balance.

`VaultLib.__callOnExternalPosition()`:

```
function __callOnExternalPosition(
    address _externalPosition,
    bytes memory _actionData,
    address[] memory _assetsToTransfer,
    uint256[] memory _amountsToTransfer,
    address[] memory _assetsToReceive
) private {
    require(
        isActiveExternalPosition(_externalPosition),
        "__callOnExternalPosition: Not an active external position"
    );

    for (uint256 i; i < _assetsToTransfer.length; i++) {
        __withdrawAssetTo(_assetsToTransfer[i], _externalPosition, _amountsToTransfer[i]);
    }

    IExternalPosition(_externalPosition).receiveCallFromVault(_actionData);

    for (uint256 i; i < _assetsToReceive.length; i++) {
        __addTrackedAsset(_assetsToReceive[i]);
    }
}

...2e42850b7bbc2237618c38fb01e767d14b606e00

    function __addTrackedAsset(address _asset) private notShares(_asset) {
    if (!isTrackedAsset(_asset)) {
        __validatePositionsLimit();

        assetToIsTracked[_asset] = true;
        trackedAssets.push(_asset);

        emit TrackedAssetAdded(_asset);
    }
}
```

The SolvV2BondBuyerPositionParser doesn't check these assets sufficiently:

```
else if (_actionId == uint256(ISolvV2BondBuyerPosition.Actions.Claim)) {
            (address voucher, uint256 tokenId, ) = __decodeClaimActionArgs(_encodedActionArgs);

            ISolvV2BondVoucher voucherContract = ISolvV2BondVoucher(voucher);

            uint256 slotId = voucherContract.voucherSlotMapping(tokenId);
            ISolvV2BondPool.SlotDetail memory slotDetail = voucherContract.getSlotDetail(slotId);

            assetsToReceive_ = new address[](2);
            assetsToReceive_[0] = voucherContract.underlying();
            assetsToReceive_[1] = slotDetail.fundCurrency;
        }
```

A voucher's underlying and fundCurrency may be any asset the IVO market supports. There is no check that the fundCurrency is an asset supported by Enyzme. If the position was bought through the external position, it's likely that the underlying is supported (else it couldn't have been bought.) Note that Solv Bond Voucher NFT positions may be transferred to an external position and consequently one cannot rely on the underlying to be supported.

Even when no value is returned in the specific asset, the asset is still added as tracked asset.

Similarly this situation may arise in the SolvV2BondIssuerPosition: `CreateOffer()` only validates that the received currency is not the native token (Ether). There is no further check on this asset which will be incoming to the vault upon `reconcile()`.

It's unclear if Enyzme supports all possible assets by default (e.g. also when new assets are added by Solv). Unsupported assets tracked by a vault may have severe consequnces as they break e.g. `calcGav()`.

It's the fund manager's responsibility to be aware and to act appropriately.