PUBLIC

Code Assessment

of the Sulu Extension XXVI Smart Contracts

October 14, 2025

Produced for

by

Contents

1 Executive Summary

Dear all,

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

Enzyme Foundation implements an external position for integrating with Alice v2 which allows for placing orders that can be taken by Alice v2 protocol.

The most critical subjects covered in our audit are asset solvency, functional correctness, integration with the external protocol, and access control. The general subjects covered are testing, gas efficiency, and trustworthiness. Security regarding the aforementioned subjects is good.

The most notable, now resolved, findings are:

- Share Price Manipulation by Triggering Hooks which illustrates how access control could have been bypassed. Additionally, the finding had implications on asset solvency.
- Incorrect Reference ID which highlighted the potential for improving functional correctness. Additionally, it illustrated how testing can be improved by mirroring the expected behavior of the external protocol more accurately.

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

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

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

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

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

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	1
• Code 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 Enzyme 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	12 Sep 2025	02c4d213313b489bcdd1dfc52ec81058b86908e5	Initial Version
2	07 Oct 2025	6fd7656b0914e7bafa0af25f13d5584fdb055b17	After Intermediate Report
3	14 Oct 2025	58163e39f6fd7257aaf58985b5945ccbfe7ac81b	Sweep Fixes

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

The files in scope were:

```
contracts/release/extensions/external-position-manager/external-positions/alice-v2/
    AliceV2PositionLib.sol
    AliceV2PositionParser.sol
    bases/AliceV2PositionLibBase1.sol
    IAliceV2Position.sol
```

2.1.1 Excluded from scope

All other files are out of scope.

Alice V2 is out of scope and expected to work correctly. Note that the code repository is private. However, access was granted. Thus, we considered commit `ab8b83d22dffe350d111860a353fdfaa27c28c28` as a reference point for the review.

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.

Enzyme Foundation implements an External Position (EP) for integration with the Alice v2 protocol.

2.2.1 External Position: Alice V2

Alice V2 Overview. Alice v2 is an order manager (`InstantOrderV2`) where whitelisted users can publish orders. Orders can be filled (settled) by Alice. Additionally, Alice or the order creator can stop orders by canceling or refunding them, respectively. Orders can also be submitted to invoke hooks upon settlement or cancellation. Specifically, the hook is invoked by providing a recipient contract and a reference ID. However, on-chain mechanisms do not guarantee hook invocation in any form.

External Position. The external position implements various actions to integrate with the above functionality. More specifically, the following actions are provided:

- `PlaceOrder` and `PlaceOrderWithRefId`: Place an order on Alice by publishing the order data and providing the necessary funds. The second action type places an order where hooks are expected to be invoked upon cancellation or settlement.
- `Sweep`: Sweeps funds received from settled or canceled orders. This action handles both "regular" orders and orders with reference IDs for which the hook was not invoked or the invocation failed.
- `RefundOrder`: Refunds an order, marking it as invalid according to Alice v2 logic. The token to be sold is transferred out, and funds from other orders could potentially be swept simultaneously.

It is important to note that there are no debt assets. The managed assets, retrieved via `getManagedAssets`, are constituted by two parts:

- For each tracked pending order: the token to be sold and its corresponding quantity.
- For each potentially sweepable token (from a tracked canceled or settled order): the EP's token balances.

Orders that include the native asset are also supported.

Furthermore, the hooks called by Alice v2 for orders with reference IDs are implemented:

- `notifySettle`: Invoked atomically after settlement with a reference ID. It automatically removes the order from storage (similar to `Sweep`) and pushes the received funds (the token to buy) to the vault proxy.
- `notifyCancel`: Invoked atomically after cancellation for an order with a reference ID. It automatically removes the order from storage (similar to `Sweep`) and pushes the received funds (the token sold) to the vault proxy.

In summary, these hooks automate the sweeping process. However, if a hook is not invoked or its invocation fails, the funds must be collected manually using the `Sweep` action.

2.3 Trust Model

This section outlines the trust model and the associated roles for the Alice V2 EP:

- **Enzyme Core**: Fully trusted and expected to work correctly. Please refer to previous audit reports for more details.
- **Fund Owner**: Fully trusted. A malicious fund owner could configure the fund to use Alice v2 to allow for draining user assets through unfavorable trades.
- **Fund Manager**: Fully trusted, depending on the fund's configuration. A fund manager could execute unfavorable trades, but the potential damage can be limited by the fund's specific configuration.
- **Alice v2**: Fully trusted to work correctly. Assuming a correct implementation, Alice v2 is partially trusted to handle orders with a reference ID. However, this review considers the potential failure of Alice v2's off-chain mechanisms.
- **Users**: Untrusted. This includes users who might maliciously integrate with Alice to trigger hooks, assuming such an interaction is possible.

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 Open 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
- 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	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1
Medium-Severity Findings	1
Low-Severity Findings	1
Informational Findings	2

- Share Price Manipulation by Triggering Hooks Code Corrected
- Incorrect Reference ID Code Corrected
- Invalid Order ID for Action Code Corrected
- Code Duplication Code Corrected
- Gas Optimizations Code Corrected

6.1 Share Price Manipulation by Triggering Hooks

Security High Version 1 Code Corrected

CS-SUL26-007

Alice v2 allows arbitrary recipient addresses and reference IDs for all order IDs. This can be leveraged to untrack orders without sweeping the correct asset to reduce the share price. Additionally, the asset could be then swept as part of another operation to allow for share price increase.

Consider the following example:

1. The EP creates a regular order USDC/WETH which is settled so that WETH is held within the EP. Assume the order ID is 100.
2. A malicious address creates a malicious order with reference with tokens USDS/USDe and specifies 100 as the reference ID.
3. `notifySettle` is invoked with `token==USDe` and `_referenceId==100`.
4. Order ID 100 is settled, so the call does not revert.
5. Order ID 100 is untracked from storage.
6. `token==USDe` is transferred out.

```
__retrieveAssetBalance({
    _asset: IERC20(_token),
    _receiver: IExternalPositionProxy(address(this)).getVaultProxy()
});
```

7. The share price decreases since the EP has no known order IDs anymore but has not received the sweepable assets.

Similarly, the correct funds could be pushed out by creating a malicious order that would lead to sweeping the WETH from the contract.

Ultimately, the share price is heavily manipulatable. However, Alice v2 allows only whitelisted users to integrate with it which limits the attack surface and thus reduces the likelihood of an attack.

Code corrected:

`notifySettle` is now independent of the `token` parameter. Additionally, only orders that were initiated with `PlaceOrderWithRefId` will be allowed in `notifySettle` and `notifyCancel`.

6.2 Incorrect Reference ID

Correctness **Medium** **Version 1** **Code Corrected**

CS-SUL26-004

Alice v2 allows users to place orders with reference IDs which leads to invocation of settlement/cancellation hooks. The EP encodes the reference ID to be the order ID so that during the hook invocation state can be cleaned up. However, an incorrect order ID is passed as the reference ID.

In particular, orders with reference ID are placed as follows:

```
ALICE_INSTANT_ORDER_V2.placeOrder{value: nativeAssetAmount}({
    ...,
    _referenceId: bytes32(ALICE_INSTANT_ORDER_V2.getMostRecentOrderId())
});
```

As visible in `__addOrder` or Alice's `placeOrder` function, the order ID created will correspond to:

```
bytes32(ALICE_INSTANT_ORDER_V2.getMostRecentOrderId() + 1)
```

To summarize, the reference ID will be unusuitable for the intended purpose. Further, note that testing does not catch this case since the "simulation" of Alice's off-chain logic ignores the event data emitted but relies on the order ID created.

Ultimately, functionality is broken, but funds can nonetheless be retrieved through "regular" operations with sweeping.

Code corrected:

The code has been adjusted accordingly.

6.3 Invalid Order ID for Action

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

CS-SUL26-003

`Sweep` accepts a list of order IDs as input from the asset manager. A lack of input sanitization, which permits duplicate order IDs (and unknown ones), creates an unexpected privilege escalation.

The internal `__sweep` function performs the following operations:

1. Enforces that an order ID has been settled or cancelled (or has never existed).
2. Removes the order ID from storage (doing nothing if the order ID is unknown).

3. Pushes out the token balances for the in- and out-tokens.

If an unknown order ID is provided, the corresponding `orderDetails` will have never been set, meaning the in- and out-tokens are `0x0`. The `parseAssetsForAction` function would report the same assets accordingly.

In contrast, if a duplicate order ID is provided, the `orderDetails` will be valid during the first iteration but will be deleted (zeroed out) before the second. Consequently, the second iteration behaves similarly to the "unknown order ID" case. However, a critical difference is that `parseAssetsForAction` will not have reported the native asset that is subsequently swept from the contract.

Consider the following scenario:

1. Assume two settled orders exist: order ID `10` for MLN/USDS and order ID `11` for USDC/ETH.
2. Consequently, the contract holds both USDS and ETH.
3. The asset manager calls `sweep` with a duplicated order ID: `[10, 10]`.
4. The parser analyzes the input and specifies that the assets to be received are MLN and USDS (from order ID `10`).
5. The first iteration correctly sweeps the USDS from the settled order.
6. The second iteration attempts to process order ID `10` again, but its state has already been deleted. This results in the native asset (ETH) being swept from the contract, even though the parser never reported it as an asset to receive.

To summarize, due to a lack of sanitization of input, ETH could be swept without reporting WETH as an asset to receive. This may ultimately lead to incorrect pricing of vault shares.

Code correct:

Enzyme Foundation disallows providing duplicate order IDs. Additionally, Enzyme Foundation ensures that the order has been created by the respective EP.

6.4 Code Duplication

Informational **Version 1** **Code Corrected**

CS-SUL26-005

`__refundOrder` implements the transfer of assets as follows:

```
// Return the refunded outgoing asset back to the vault
IERC20 outgoingAsset = IERC20(orderDetails.outgoingAssetAddress);

if (address(outgoingAsset) == ALICEV2_NATIVE_ASSET_ADDRESS) {
    Address.sendValue(payable(msg.sender), address(this).balance);
} else {
    outgoingAsset.safeTransfer(msg.sender, outgoingAsset.balanceOf(address(this)));
}
```

However, the function `__retrieveAssetBalance` provides the equivalent functionality:

```
function __retrieveAssetBalance(IERC20 _asset, address _receiver) private {
    uint256 balance =
        address(_asset) == ALICEV2_NATIVE_ASSET_ADDRESS ? address(this).balance : _asset.balanceOf(address(this));

    if (balance > 0) {
```

```
        // Transfer the asset
        if (address(_asset) == ALICEV2_NATIVE_ASSET_ADDRESS) {
            Address.sendValue(payable(_receiver), balance);
        } else {
            _asset.safeTransfer(_receiver, balance);
        }
    }
}
```

Note that the result is the same. Using `__retrieveAssetBalance` could lead to better readability and consistency in the EP's code.

Code corrected:

The code duplication has been removed.

6.5 Gas Optimizations

Informational | Version 1 | Code Corrected

CS-SUL26-006

Below is a list of potential gas optimizations:

1. `__refundOrder`: The function gets the order details with `getOrderDetails`. However, that is not strictly needed since `refundOrderArgs` contains the `outgoingAssetAddress` as `refundOrderArgs.tokenToSell`.

Code corrected:

The order details are not used.

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 Manipulating `__isOrderSettledOrCancelled`

Informational **Version 1** **Risk Accepted**

CS-SUL26-001

Governance should be aware that the function `__isOrderSettledOrCancelled` could be subject to manipulations. This could result in incorrect execution.

Specifically, the gas provided to the call could be manipulated so that the function returns an incorrect boolean. Consider the function's implementation:

```
function __isOrderSettledOrCancelled(uint256 _orderId) private view returns (bool isSettledOrCancelled_) {
    // When an order has been settled or cancelled, its orderHash getter will throw
    try ALICE_INSTANT_ORDER_V2.getOrderHash({_orderId: _orderId}) {
        return false;
    } catch {
        return true;
    }
}
```

In case of reverts, the `try/catch` returns `true`. Note that the same is the case for out-of-gas reverts. Hence, the function could return `true` more often than it actually is. If the order is still pending but such a revert occurs, the EP would assume that the position has been settled. This would lead to incorrectly continuing the execution in `notifySettle`, `notifyCancel` and `__sweep`. Additionally, `getManagedAssets` may use the incorrect path for the execution.

Note that such manipulations, given the context of the EP and the current cost of the operations are unlikely and no example has been found. However, changes in gas prices may facilitate potential attacks.

Note that the integrated with function reverts with the reason:

```
"Invalid Order Id / Order already settled."
```

Any other reason should typically be invalid as it is not possible to determine in such cases whether an order has been settled or cancelled. Hence, the code should revert if an unexpected revert reason is returned.

Risk accepted:

Enzyme Foundation is aware and accepts the risk.

7.2 Parser Overreliance on Alice V2 Off-Chain Mechanism

Informational **Version 1** **Risk Accepted**

CS-SUL26-002

Alice V2's off-chain mechanism is undocumented but is expected to detect order with references and execute settlement and cancellation in accordance with that. However, the EP's parser may rely too much on the correctness of the off-chain mechanism.

More specifically, there is an inconsistency between the `assetsToReceive_` for actions `PlaceOrder` and `PlaceOrderWithRefId`. That is generally expected to be fine as for the former as the assets would be reported as part of `Sweep`. However, in case of problems in Alice's v2 off-chain mechanism, it could hypothetically happen that for `PlaceOrder` as settlement with reference ID occurs. As such, the assets would not be tracked.

Consider the following example:

1. An order is placed where WETH is being sold and USDC is being bought.
2. The parser has not reported any assets being received.
3. Alice has a bug in the off-chain mechanism and incorrectly invokes `settleOrderWithReference`.
4. The vault proxy received USDC but does not know that it holds USDC.
5. Ultimately, share price drops.

Risk accepted:

Enzyme Foundation is aware and accepts the risk.

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 Alice V2 Integration Considerations

Note Version 1

Below is a list of considerations fund owners and managers, governance and users should be aware of:

1. `notifySettle` / `notifyCancel` could be used to allow arbitrary sweeping of unswept funds. However, generally this does not lead to problems. Note that the reason for that is that Alice V2 allow anyone to specify arbitrary reference IDs and hook recipients.
2. There are no guarantees that offers with or without reference ID are executed accordingly.
3. The whitelisted tokens in Alice v2 are expected to be non-reentrant and non-special tokens. Enzyme Foundation should monitor the supported tokens in Alice v2 as they could introduce unexpected attack vectors.

As of Version 2, 1. has been restricted further.