PUBLIC

Code Assessment

of the Sulu Extension XXV Smart Contracts

August 22, 2025

Produced for

enzyme

by

CHAINSECURITY

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

Enzyme Foundation updates the GMXv2 external position to support the upcoming update from v2.1 to v2.2. The main changes are the addition of a new hook entrypoint and a change in how positions are priced.

The most critical subjects covered in our audit are functional correctness, asset solvency, and Enzyme's integration with the external system. The integration problems Claimable collateral can be overestimated and Valuation Inconsistencies have been addressed and resolved accordingly.

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	0
Low-Severity Findings	2
• Code Corrected	2

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	11 Aug 2025	929f3a97cf2ffefda81040d92ceb1a6a4a284503	Initial Version
2	19 Aug 2025	3b9d8537dd1b865d8321d1b8ef7344a7d957fad4	After Intermediate Report
3	22 Aug 2025	70240466d5ab8b2b31f46b07347820b4817f5d6b	Further fixes

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

The scope of the assessment includes the following files:

```
contracts/release/extensions/external-position-manager/external-positions/gmx-v2-leverage-trading/
    bases/
        GMXV2LeverageTradingPositionLibBase1.sol
    GMXV2LeverageTradingPositionLib.sol
    GMXV2LeverageTradingPositionLibManagedAssets.sol
    GMXV2LeverageTradingPositionMixin.sol
    GMXV2LeverageTradingPositionParser.sol
    IGMXV2LeverageTradingPosition.sol
```

2.1.1 Excluded from scope

Any contracts inside the repository that are not mentioned in `Scope` are not part of this assessment. All external libraries, and imports are assumed to behave correctly according to their high-level specification, without unexpected side effects.

The correctness of external systems is not in scope.

Tests and deployment scripts are excluded from the scope.

The main part of this review took place in the first half of August 2025; the review is based on the state of GMX v2.2 at that time. Any changes to the code after this date are not considered in this report. Further, we expect the following that the migration is handled carefully. Optimally, the EP will be updated directly after GMX's upgrade (and could potentially be paused shortly before the update to prevent `getManagedAssets` from potentially returning incorrect values temporarily).

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 updates the GMXv2 leverage trading external position to support the upgrade of the GMX from v2.1 to v2.2.

GMX is a decentralized perpetual exchange that allows users to trade various assets with leverage on-chain. Version 2.2 introduces several improvements and new features. While we considered all of them in our assessment, we will focus in this overview on the changes that are relevant to the external position (EP). For the complete system overview, please refer to the initial report.

Callbacks have been modified and now take an *EventUtils.EventLogData* struct instead of an *Order.Props* as an argument. Furthermore, during the transition period from v2.1 to v2.2, both callback signatures must be supported. Thus, the `afterOrderExecution()` function was refactored, and two overloaded versions were introduced. The first version supports the old callback signature, and the second version supports the new callback signature. The general logic of the callback was moved to the internal `__afterOrderExecution` function, which is called by both overloaded versions.

GMX v2.2 modifies when the price impact is charged to a user. Previously, the price impact was charged at the time of the order execution for both increase and decrease orders. In v2.2, the price impact on increase orders is stored in the position and charged when the position is decreased. As the EP aims at pricing the open positions as if they were closed at the current price, the new price impact logic should be taken into account when calculating the position value. Thus, the logic computing the collateral of a position in `getManagedAssets()` was modified to account for the `totalImpactUsd` which is the sum of the price impact of opening and closing the position. Moreover, positions opened prior to version 2.2 and priced by using data from `ReaderPositionUtilts` in v2.2 can still be correctly priced. The position will already have paid for the price impact on decrease and thus `pendingImpactAmount` should be 0. Which will be case as uninitialized storage will be read, returning 0, when fetching `pendingImpactAmount` and thus `totalImpactUsd` will only represent the decrease impact for such positions.

Additionally, `getManagedAssets()` was modified to include funding fees not registered within storage yet. More specifically, `claimableLongTokenAmount` and `claimableShortTokenAmount` which are the claimable amounts of long and short tokens in the position accrued from funding. These will be made claimable by the user when the position is touched by the user (increased or decreased) or when the position is liquidated.

The rest of the changes are related to supporting the new interfaces and return structures introduced in GMX v2.2, such as the new *OrderInfo* struct which wraps an *Order.Props* and the *orderKey*, thus one additional level of indirection is introduced when accessing `getAccountOrders()` return value.

For the full change log of GMX, please refer to this link.

2.3 Trust Model

Please refer to the main code assessment report and the extension reports for a general trust model of Sulu.

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

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.

For the GMX EP, both the fund owner and the asset managers are fully trusted. For example, unreasonable positions leading to liquidations and fund loss could be created. Similarly, orders with no effect could be created to keep paying execution fees. Alternatively, one can take part in manipulations leading to reductions of claimable factors and thus to potential share price arbitrage.

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. External systems (i.e. GMX) are expected to work correctly and as expected.

GMX governance is fully trusted and expected to not manipulate the EP by changing configuration or account parameters or the logic of the GMX contracts in a way that would lead to a loss of funds or a bricking of the EP.

Further, it is assumed that the fund manager only acts through Enzyme's official interface for GMX. The CHAINLINK_PRICE_FEED_PROVIDER in the EP is assumed to be set to `ChainlinkPriceFeedProvider`, a custom implementation by GMX. The price returned is assumed to be the price of one unit of the token using a value with 30 decimals of precision. This allows for conversions between token amounts and fiat values to be: token amount * oracle price, to calculate the token amount for a fiat value: fiat value / oracle price. For more details please refer to the examples provided by GMX.

Only valid exchange routers are expected to have the CONTROLLER_ROLE and have the functions called by the EP accessible. While other contracts may hold this role, e.g. the `OrderHandler` calls from the EP are expected to fail due to the function selector not being available / the function not being accessible. Hence, checking `__assertHandler` is assumed to be sufficient to ensure that the EP is interacting with a valid GMX exchange router. The GMX Pools that the EP interacts with are expected to have sufficient liquidity; it is assumed they cannot be manipulated (price impact).

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:

- 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	0
Medium-Severity Findings	0
Low-Severity Findings	2

- Claimable Collateral Can Be Overestimated Code Corrected
- Valuation Inconsistencies Code Corrected

6.1 Claimable Collateral Can Be Overestimated

Correctness Low Version 1 Code Corrected

CS-SUL25-001

`getManagedAssets()` calculates the total value of all positions that the GMXv2 external position holds. This includes taking into account the collateral that is claimable by the external position on GMX.

The claimable collateral is computed as the difference between the collateral that is claimable and the collateral that has already been claimed.

In GMXv2, `claimCollateral()` applies a `claimableFactor` to adjust the amount of claimable collateral.

```
uint256 claimableFactor = _getClaimableFactor(dataStore, market, token, timeKey, account);

if (claimableFactor > Precision.FLOAT_PRECISION) {
    revert Errors.InvalidClaimableFactor(claimableFactor);
}

uint256 claimedAmount = dataStore.getUint(Keys.claimedCollateralAmountKey(market, token, timeKey, account));

uint256 adjustedClaimableAmount = Precision.applyFactor(claimableAmount, claimableFactor);
```

This factor is not applied when calculating the claimable collateral in `getManagedAssets()` in the external position. Note that the factor is enforced to be between 0 and `FLOAT_PRECISION` which in this case is 10^30, thus the amount of claimable collateral can only be overestimated, not underestimated.

Code corrected:

The code has been corrected. The claimable factor is now included in the computations.

6.2 Valuation Inconsistencies

Correctness Low Version 1 Code Corrected

CS-SUL25-002

The valuation of an EP is defined within `GMXV2LeverageTradingPositionLibManagedAssets.getManagedAssets`. It consists of multiple factors as outlined in NatSpec. The computations regarding point 1, the collateral of GMX positions, aims to simulate a full decrease of the GMX positions to estimate the value. However, `getManagedAssets` does not replicate the logic consistently.

Below is a simplification `DecreasePositionCollateralUtils.processCollateral`:

```
out = 0

if basePnlUsd > 0:
    out += basePnlUsd / price.max
else if basePnlUsd < 0:
    out -= divUp(-basePnlUsd, price.min)

if values.totalImpactUsd > 0:
    out += totalImpactUsd / price.max
else if totalImpactUsd < 0:
    out -= divUp(-totalImpactUsd, price.min)

out += position.collateralAmount
out -= totalCostAmount
```

Note that only relevant parts are highlighted, execution is reordered, `totalCost` is used instead of individually deducting `totalCostAmountExcludingFunding` and `fundingFeeAmount`. Further, note that the above excludes logic where GMX is at loss (capping value to be at least 0) and also does not mention the logic related to `priceImpactDiffUsd` as the effective delta on the assets is 0.

The EP, in contrast, defines the valuation of a GMX position as:

```
out = 0

netPnlUsd = basePnlUsd + totalImpactUsd
if netPnlUsd > 0:
    out += netPnlUsd / price.max
else if netPnlUsd < 0:
    out -= -netPnlUsd, price.min

out += position.collateralAmount
out -= totalCostAmount
```

Note that the two implementations are inconsistent. Thus, the following multiple problems are created:

1. Assume that `basePnlUsd` is negative and that `totalImpactUsd` is positive. Then, GMX would compute `divUp(-basePnlUsd, price.min)` and `totalImpactUsd / price.max` as part of the sum. However, the EP would either compute `(basePnlUsd + totalImpactUsd) / price.max` or the corresponding version with `min` depending on the sign of `netPnl`. Hence, the EP's valuation might not be detailed enough as `price.min` or `price.max` could potentially be used incorrectly.
2. The EP does not consider the `divUp` in any case which can lead to minor rounding errors.

To summarize, `getManagedAssets` does not replicate the GMX logic consistently.

Code corrected:

`getManagedAssets()` was updated to replicate the logic of `DecreasePositionCollateralUtils.processCollateral`.

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 Considerations for Claimable Factor in GMX

Note Version 1

The claimable factor determines the share of claimable collateral a user on GMX can receive. The mechanism is roughly outlined in the GMX documentation. The relevant code can be found in GMX's codebase.

Due to the mechanism, various corner cases are created. Below is a list of the most relevant ones for the EP:

1. *Reverts due to decreasing the claimable factor*. For example, if a factor of 50% is set and collateral is claimed, and the factor is then decreased to 25%, reverts can occur in the subtraction of "adjusted claimable amount" and "claimed amount".
2. *Factor changes lead to share price jumps*. For example, if a factor is changed, the share price will be immediately affected (i.e. increases/decreases in factor lead to increases/decreases, respectively).
3. *Price-impact differences*. Note that the claimable collateral corresponds to `priceImpactDiffUsd`. However, that is only realized when a position is touched. Thus, depending on the time of the actual execution, a penalty could be present. Enzyme Foundation will not consider such penalties as they are unpredictable.

Further, note that after discussion with GMX and Enzyme Foundation, the following assumptions have been specified:

1. The two relevant claimable collateral factors are typically not expected to be set anymore.
2. The newly introduced (as of GMX v2.2) claimable reduction factor is expected to be set.
3. The claimable factor is expected to be set once. Potentially, it could be set a second time after discussion of the penalty. However, that is not be assumed by default.
4. EPs are typically expected not be penalized and are typically expected to have 100% shares in their claimable collateral normally.

Note that the following holds for the EP:

1. Considers the claimable factor but defaults to 100% which is the expected value when nothing has been written to storage on GMX or when it is unpredictable.
2. Further, it assumes that GMX will act responsibly in terms of setting the factors (e.g. no decrease after claim occurred, no manipulations).
3. Additionally, the valuation of the EP is directly affected by factor changes as they are generally expected not be changed in the future.
4. Last, in case an EP has been penalized by GMX, the "claimable collateral key" will remain tracked in case there is an increase.