PUBLIC

Code Assessment

of the Sulu Extensions XVIII Smart Contracts

August 22, 2024

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

Enzyme Foundation implements a new external position for GMX V2. Supported orders are `MarketIncrease`, `MarketDecrease` (to modify long or short positions), `StopLossDecrease` (to set stop loss) and `LimitDecrease` (to set take profit). Additionally, a new policy `DisallowedAdapterIncomingassetsPolicy` has been implemented.

The most critical subjects covered in our audit are functional correctness, asset solvency and Enzyme's integration with the external system.

Functional correctness did not hold due to claimable collateral that might have been tracked incorrectly. For details please refer to the issue: Overestimation of Claimable Collateral in getManagedAssets. Further, the position's value could have been temporarily decreased by hiding value in the execution fee, which was not accounted for when evaluating the external position's total value: ExecutionFee of Orders in getManagedAssets.

After the intermediate report, all issues have been resolved.

During the assessment period, it became apparent that the technical documentation for GMX V2 lacks key information. As a result, our in-depth understanding of the external system was primarily derived from analyzing the available source code.

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	1
• Code Corrected	1
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	3
• Code Corrected	3

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 Protocol repository. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	12 August 2024	81fb006d67f0c0b923e2f7ebb9234e8e7c870d3f	Initial Version
2	20 August 2024	0f132460d4c5018327027582728f56348b288aa6	After Intermediate Report

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

The scope for the GMX V2 Protocol external position is:

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

In Version 2, the scope has been updated to include the DisallowedAdapterIncomingAssetsPolicy:

```
contracts/release/extensions/policy-manager/policies/asset-managers/DisallowedAdapterIncomingAssetsPolicy.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 2024; the review is based on the state of GMX V2 at this point in time. This review cannot account for future changes.

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 a new external position for integrating with GMX V2 on Arbitrum.

2.2.1 External Position: GMX

GMX is a decentralized perpetuals exchange that supports leverage trading. It offers both fully backed as well as synthetic markets where users can open long/short positions. The external position (EP) enables Enzyme vaults to trade on GMX. For a detailed description of how external positions function, please refer to the main code assessment report of Sulu.

The external position implements the following actions:

1. `CreateOrder`: Creates a new order on GMX. The implementation supports one of the following `OrderTypes`: `MarketIncrease`, `MarketDecrease`, `StopLossDecrease`, `LimitDecrease`. Please find a brief description of each of these order types towards the end of this section.
2. `UpdateOrder`: Updates parameters of existing orders. Since GMX does not allow to update market orders, this action can only be applied to stop-loss and limit decrease orders.
3. `CancelOrder`: Cancels a pending order and transfers the remaining execution fees to the vault. If the canceled order is of type `MarketIncrease` the associated collateral is also transferred back.
4. `ClaimFundingFees`: Claims positive funding fees for given market-token pairs from the GMX protocol.
5. `ClaimCollateral`: Claims claimable collateral for the specified market-token-timekey tuples, the collateral is returned to the vault.
6. `Sweep`: Transfers tokens of the tracked assets held by the external position to the vault. Also removes stale tracked asset by rebuilding the tracked assets array based on the active `MarketIncrease` orders and current positions.

Orders on GMX are executed in two steps: The user creates an order which will be executed in a second transaction by a keeper.

The external position with GMX does not take on debt. The amount and types of assets managed by the vault's external GMX position can be retrieved via `getManagedAssets()`. Note that the total value of assets managed by the external position is composed of several factors:

- The valuation of collateral that is held in active GMX positions. It is computed by using the conservative bound of the oracle price range and hence under-estimates the actual value.
- The value of the `MarketIncrease` orders that are pending (have not been executed yet).
- The value of assets held by the external position due to refunds of execution fees, liquidations or cancellation of pending orders.
- The value of released collateral from decreased GMX positions.
- Positive funding fees that can be claimed from the GMX protocol.

We briefly describe the four possible order types implemented in the external position for GMX:

1. `MarketIncrease`: A market order to increase or open a position (long or short); transfers the collateral (and execution fee) to the exchange router.
2. `MarketDecrease`: A market order to decrease or close a position (long or short); transfers execution fee to the exchange router.
3. `StopLossDecrease`: A stop-loss order to decrease a position (long or short); should be executed according to the trigger price.
4. `LimitDecrease`: A limit order to decrease a position (long or short); should be executed according to the trigger price.

Callers specify parameters in `CreateOrderActionArgs`, which are then used to generate `CreateOrderParams` for interaction with GMX. Note that not all functionalities are accessible; notably, swaps prior to the order have been disabled. This enforces the vault to send long and/or short tokens from the respective market as collateral.

For each market the vault is participating in, the vault's external position is set as callback contract for GMX upon the first creation of a `MarketIncrease` order. This stored callback contract is used for async actions such as liquidations and ADL in GMX. The callback address for orders is set within the order parameters as the vault's external position proxy or `0x0` for `MarketIncrease` orders. The EP provides a callback function `afterOrderExecution` that is called by GMX when a decrease order has been executed and in the case of liquidation. The callback function updates `claimableCollateralKeys` and `claimableCollateralKeyToClaimableCollateralInfo` if additional collateral has been released from the order execution due to the order exceeding the negative price impact threshold. Thus, the EP keeps track of released collateral that still needs to be claimed with `__claimCollateral()`.

GMX uses the concept of time keys to group users' claimable collateral in the protocol accounting. The current time slot divisor is set to 3600 (1h) in GMX, which implies that the `timeKey` has a different value for every hour. Therefore, a user's claimable collateral is stored in the GMX `DataStore` under the respective market, account address and time key of when the collateral was released. Thus, the EP keeps track of the time keys in addition to the market and account address to claim all collateral.

Changes in Version 2

A new policy, DisallowedAdapterIncomingAssetsPolicy, has been implemented to check for disallowed assets after calls on integrations. This policy passes if none of the incoming assets are present in any of the configured disallowed lists.

2.2.2 Roles and 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.

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.

Further, it is assumed that the fund manager only acts through Enzyme's official interface for GMX. The `CHAINLINK_PRICE_FEE_PROVIDER` 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 external position accessible. While other contracts may hold this role, e.g. the OrderHandler calls from the external position 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 external position 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 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	0

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
Medium-Severity Findings	1
Low-Severity Findings	3
Informational Findings	1

- High: ExecutionFee of Orders in getManagedAssets — Code Corrected
- Medium: Overestimation of Claimable Collateral in getManagedAssets — Code Corrected
- Low: Callback Contract Stored Is Independent of Router — Code Corrected
- Low: Incorrect Assumption for Claimed and Claimable Collateral — Code Corrected
- Low: UpdateOrder Does Not Support to Increase the Fee Amount — Code Corrected
- Informational: Inaccurate Description — Specification Changed

6.1 ExecutionFee of Orders in getManagedAssets

Correctness **High** **Version 1** **Code Corrected**

CS-SUL18-001

Pending `MarketIncrease` orders contribute to the valuation of the external position, such orders have a collateral amount and execution fee associated.

The collateral value is accounted for in `getManagedAssets()`:

```
// 2. Get pending market increase orders value
IGMXV2Order.Props[] memory orders = __getAccountOrders();

for (uint256 i; i < orders.length; i++) {
    IGMXV2Order.Props memory order = orders[i];
    if (order.numbers.orderType == IGMXV2Order.OrderType.MarketIncrease) {
        assets_ = assets_.addItem(order.addresses.initialCollateralToken);
        amounts_ = amounts_.addItem(order.numbers.initialCollateralDeltaAmount);
    }
}
```

This does not account for the execution fee which has been provided alongside the order. Excess execution fee is refunded to the external position after the order has been executed; this e.g. allows to hide value as execution fee which will not be included in the valuation thereby leading to a lower valuation returned by `getManagedAssets()` temporarily before the excess fee is returned.

The `getManagedAssets` value could be manipulated by placing a `MarketIncrease` order with a high execution fee, then buying vault shares at a reduced price and finally canceling the order.

Similarly, execution fees of other pending orders (`MarketDecrease`, `StopLossDecrease` and `LimitDecrease`) are also not taken into account.

Code corrected:

The valuation of the managed assets now includes execution fees for pending orders.

6.2 Overestimation of Claimable Collateral in getManagedAssets

Correctness | Medium | Version 1 | Code Corrected

CS-SUL18-002

`getManagedAssets()` relies on the `claimableCollateralKeys` array in step 4 to determine the amount of collateral that can be claimed.

`claimableCollateralKeys` is populated with the keys of the collateral that can be claimed in the `afterOrderExecution()` callback if the executed order resulted in collateral being claimable. The keys added to `claimableCollateralKeys` are `DATA_STORE` keys that represent the aggregated amount of collateral that became claimable on a `market` for a `token` in the period `timeKey`. In the current deployed implementation, the period is 1 hour.

However, multiple orders can be executed in the same period, on the same market, for the same token that exceed the negative price impact threshold. Therefore, the same key could be added to `claimableCollateralKeys` multiple times. This will cause `getManagedAssets()` to double count the amount of collateral that can be claimed.

```
// __addClaimableCollateral()
bytes32 key = __claimableCollateralAmountKey({_market: _market, _token: _token, _timeKey: _timeKey});

claimableCollateralKeys.push(key); // Can push the same key multiple times
```

Code corrected:

`__addClaimableCollateral()` now ensures that a key is added to `claimableCollateralKeys` only once.

6.3 Callback Contract Stored Is Independent of Router

Design | Low | Version 1 | Code Corrected

CS-SUL18-003

A nested mapping per router and market is used in the external position to track whether a callback contract is set: `exchangeRouterToMarketToIsCallbackContractSet[_exchangeRouter][_market]`. GMX however only stores the callback contract per account and market, independent of the ExchangeRouter:

```
function setSavedCallbackContract(DataStore dataStore, address account, address market, address callbackContract) external {
    dataStore.setAddress(Keys.savedCallbackContract(account, market), callbackContract);
}
```

The current implementation of the external position might give the impression that a callback contract is set for each market and router pair, which isn't the case. Since the external position always sets itself as the callback, there's no risk of error. The only impact is a slight increase in gas cost due to writing additional mapping entries.

Code corrected:

The callback is now stored per market only, the mapping used has been changed to `marketToIsCallbackContractSet[_market]`. Additionally a public getter `getMarketToIsCallbackContractSet()` has been introduced for accessing the internal mapping.

6.4 Incorrect Assumption for Claimed and Claimable Collateral

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

CS-SUL18-004

In GMX, `MarketUtils.claimCollateral()` computes the amount of collateral that can be claimed by an account for a market. As the claimable collateral can be released over time, a `claimableFactor` exists to determine the amount of collateral that can be claimed at a given time. An additional check is then performed that `adjustedClaimableAmount > claimedAmount`. Therefore, GMX remains conservative by handling the unlikely case where the claimed collateral amount exceeds the claimable collateral amount.

However, the current EP implementation implicitly assumes that the claimable collateral amount is either equal to or greater than the claimed collateral amount. This assumption is not guaranteed by the GMX implementation.

In `__claimCollateral()` of the EP, the following code snippet is used to check if all the collateral was claimed:

```
// check if all the collateral was claimed
if (claimableCollateral - claimedCollateral == 0) {
```

Thus, the case where `claimableCollateral < claimedCollateral` is not handled correctly and will lead to a revert due to underflow.

Similarly, in `getManagedAssets()` of the EP, the following code snippet is used to compute the remaining collateral that can be claimed.

```
amounts_ = amounts_.addItem(
                DATA_STORE.getUint(key)
                    - DATA_STORE.getUint(
                        __claimedCollateralAmountKey({
                            _market: claimableCollateralInfo.market,
                            _token: claimableCollateralInfo.token,
                            _timeKey: claimableCollateralInfo.timeKey
                        })
                ) // claimable collateral - claimed collateral
            );
```

Here too, the case where `claimableCollateral < claimedCollateral` is not handled correctly and will lead to a revert due to underflow. Therefore, `getManagedAssets()` would not be able to be

called to retrieve the amount of assets managed by the external position to compute the value of a vault share.

Due to the potential unexpected reverts, it would then require the fund manager to wait until `claimableCollateral` becomes greater or equal than `claimedCollateral` to execute `claimCollateral()`. The same holds for anyone who wants to call `getManagedAssets()`.

Code corrected:

The code has been changed and now follows GMX's conservative approach in `_claimCollateral()`. Hence the unlikely case where the claimed collateral amount exceeds the claimable collateral amount is now handled. `getManagedAssets()` remains unchanged. Since the amount of claimed collateral only changes in `__claimCollateral()`, the claimable collateral is only increasing and the key is removed in `__claimCollateral()` if there is no more collateral, the subtraction in `getManagedAssets()` cannot underflow anymore.

6.5 UpdateOrder Does Not Support to Increase the Fee Amount

Design | Low | Version 1 | Code Corrected

CS-SUL18-005

When updating an order in GMX, it's possible to top up the execution fee for the order by transferring additional WNT. This is mainly meant to be used for frozen orders. However, the `UpdateOrder` action allows to update the params of the order only but does not feature support to increase the fee amount. Consequently, updating a frozen order, hence unfreezing an order, will fail as it requires an additional execution fee to be sent alongside it.

Action `UpdateOrder` now supports to increase the execution fee.

6.6 Inaccurate Description

Informational | Version 1 | Specification Changed

CS-SUL18-006

The NatSpec of `getManagedAssets()` roughly describes the contributions to the valuation of the external position. For positions, the description is not accurate:

```
// 1. Collateral in the GMX positions
```

The value of the position is calculated in terms of collateral taking into account price impact and profit or loss.

Specification changed:

The description has been updated.

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 Valuation of Claimable Collateral

Note Version 1

`getManagedAssets()` takes into account the full value of the released collateral to be claimed. Note that while this collateral belongs to the external position, as it's released over time not the full amount might be claimable at this point in time.

The comment:

```
4. Get the value of the collateral that can be claimed from GMX
positions where collateral was freed up from decreased positions.
```

is slightly inaccurate; while it's collateral belonging to the external position and can eventually be claimed; not all of it may be claimable at this point in time.

The code has been annotated to clarify that this collateral can eventually be claimed from GMX.