Scanned Code Report

AUDITAGENT

Powered by NETHERMIND SECURITY

Code Info

Developer Scan

Scan ID 13	Date March 01, 2026
Organization RigoBlock	Repository v3-contracts
Branch feat/perps	Commit Hash 2bdc52ef...d855122d

Contracts in scope

contracts/protocol/libraries/GmxLib.sol

Code Statistics

Findings 0	Contracts Scanned 1	Lines of Code 314

Findings Summary

Total Findings

- High Risk (0)
- Medium Risk (0)
- Low Risk (1)
- Info (0)
- Best Practices (1)

Code Summary

This contract is a Solidity library, `GmxLib`, that serves as a specialized utility for interacting with the GMX v2 perpetuals protocol on the Arbitrum network. It is designed to be used internally by other contracts to read and interpret a user's GMX position data. The library encapsulates the logic for calculating the total value of a user's assets within GMX, including both active positions and pending orders, and presents it in a simplified `AppTokenBalance` format.

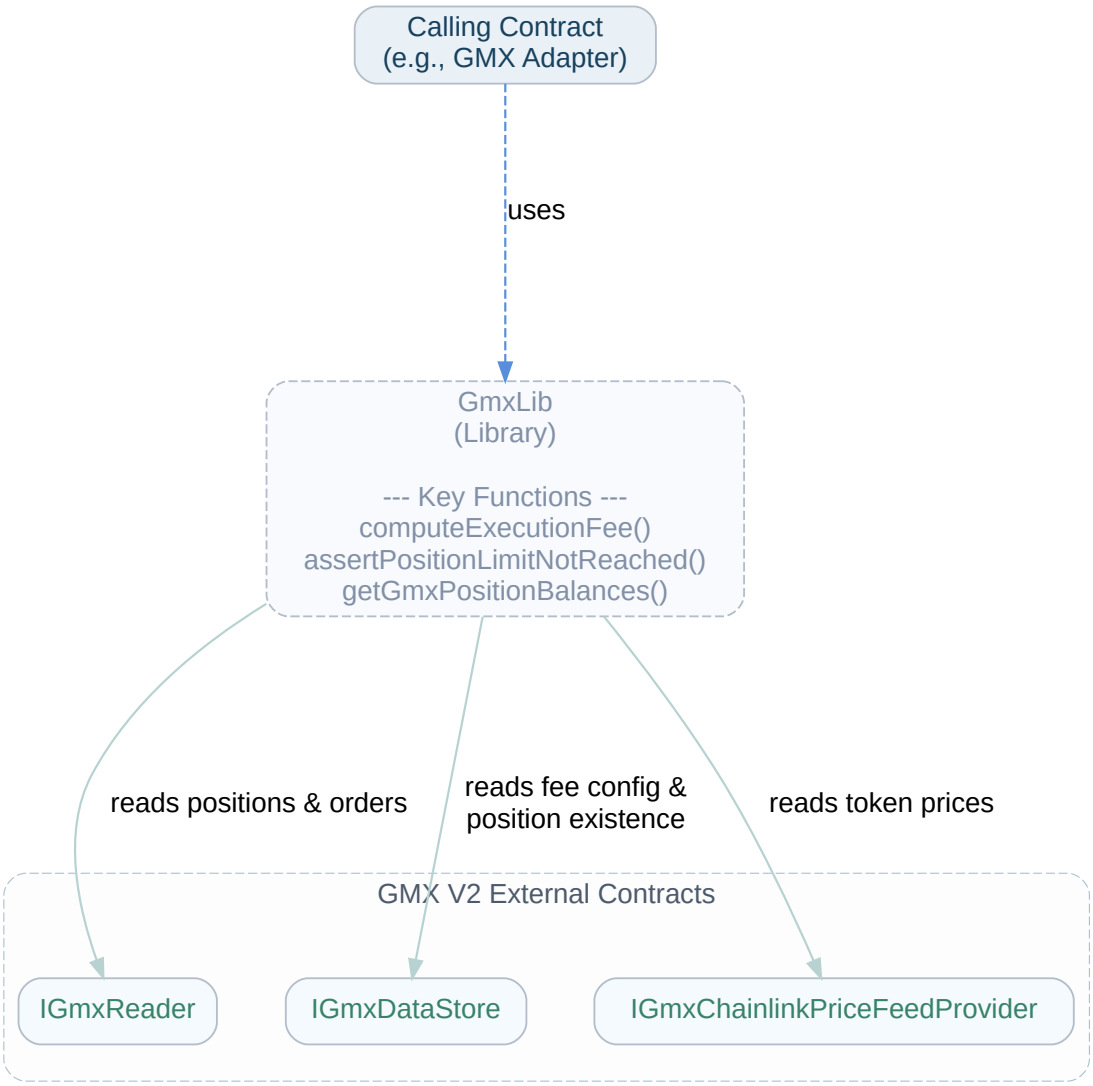
Key functionalities of the library include:
- **Executed Position Valuation:** It calculates the net value of a user's open positions by fetching the collateral amount, unrealized profit and loss (PnL), price impact, and any accrued fees from GMX's contracts.
- **Pending Order Valuation:** It accounts for the value of collateral and execution fees locked for pending increase orders that have not yet been executed.
- **Position Limit Checks:** It provides a helper function, `assertPositionLimitNotReached`, to verify that creating a new position will not exceed the maximum number of positions allowed per user on GMX.
- **Data Aggregation:** It aggregates all components of a user's GMX holdings, including net collateral and claimable funding fees for different tokens, into a single, comprehensive array.

Architecturally, `GmxLib` functions as a read-only adapter, containing hardcoded addresses for key GMX v2 contracts and abstracting away the complexity of direct interaction with them.

Entry Points and Actors

This contract is a library and does not contain any direct, state-modifying public entry points. It provides internal helper functions intended to be used by other contracts within the protocol to read data from GMX v2.

Code Diagram



1 of 2 Findings

contracts/protocol/libraries/GmxLib.sol

Pending increase orders with zero collateral cause executionFee (WETH) to be omitted from NAV / AppTokenBalance output

Low Risk

The pending-order valuation path in `GmxLib` can silently omit the value of execution fees that are locked in GMX for certain pending **increase** orders.

Where it happens

In `_getPendingOrderBalances`, the loop skips any qualifying increase order whose `initialCollateralDeltaAmount` is zero:

```
Order.OrderType ot = orders[i].order.numbers.orderType;
if (ot != Order.OrderType.MarketIncrease && ot != Order.OrderType.LimitIncrease) continue;

address colToken = orders[i].order.addresses.initialCollateralToken;
uint256 amount = orders[i].order.numbers.initialCollateralDeltaAmount;
if (amount == 0) continue;

// ... collateral entry appended ...

uint256 fee = orders[i].order.numbers.executionFee;
if (fee > 0) {
    tmp[count++] = AppTokenBalance({token: WRAPPED_NATIVE, amount: int256(fee)});
}
```

Because of the early `continue`, the execution fee is only appended when `initialCollateralDeltaAmount > 0`.

Why this can be wrong in practice

GMX orders can be considered non-empty even if `initialCollateralDeltaAmount == 0` as long as other fields (e.g., `sizeDeltaUsd`) are non-zero. GMX's order validation (as seen in deployed handler code) treats an order as empty only if **both** `sizeDeltaUsd == 0` and `initialCollateralDeltaAmount == 0`. (ww4.arbiscan.io)

This means it is possible to have a pending `MarketIncrease` / `LimitIncrease` order where:
- `orderType` is an increase type,
- `initialCollateralDeltaAmount == 0` (no collateral moved),
- `executionFee > 0` (WETH/WNT still transferred and locked for keeper execution).

In that case, this library returns **no AppTokenBalance entry** for the locked execution fee, understating the account's total GMX-held value.

Impact

If the protocol relies on `getGmxPositionBalances()` output as part of NAV/share pricing:
- NAV is understated while such orders are pending.
- Understated NAV can translate into incorrect mint/redeem/share calculations (dilution effects): participants transacting while NAV is understated can receive a more favorable share price than they should, at the expense of other LPs.

This is specifically a correctness / accounting issue in the adapter’s read-path: the execution fee is value that is already paid by the account and remains attributable to the account (refunded on cancellation / partly spent on execution), but it is not reflected in the returned balances for the pending window.

Severity Note:
- The protocol uses getGmxPositionBalances() (or its outputs) directly in NAV/share pricing for mint/redeem.
- Zero-collateral increase orders are placed by the strategy with non-zero executionFee on GMX v2.
- Order execution is not instantaneous, leaving a non-zero window where deposits can be processed against the understated NAV.

2 of 2 Findings

contracts/protocol/libraries/GmxLib.sol

Unsafe uint256 to int256 casts could cause integer overflow and incorrect accounting

Best Practices

Multiple functions perform unchecked casts from `uint256` to `int256` for token amounts and collateral values. If any of these uint256 values exceed `type(int256).max` (2^255 - 1), the cast will silently wrap to a negative value, causing severe accounting errors in NAV calculations.

Affected locations:

1. In `_appendGmxPosBalances`:

```
uint256 cl = posInfo.fees.funding.claimableLongTokenAmount;
if (cl > 0) {
    tmp[count++] = AppTokenBalance({token: mkt.longToken, amount: int256(cl)});  // Unsafe
cast
}
uint256 cs = posInfo.fees.funding.claimableShortTokenAmount;
if (cs > 0) {
    tmp[count++] = AppTokenBalance({token: mkt.shortToken, amount: int256(cs)});  // Unsafe
cast
}
```

2. In `_computeGmxNetCollateral`:

```
netCollateral =
    int256(posInfo.position.numbers.collateralAmount) +  // Unsafe cast
    basePnlCollateral +
    impactCollateral -
    int256(posInfo.fees.totalCostAmount);  // Unsafe cast
```

3. In `_getPendingOrderBalances`:

```
tmp[count++] = AppTokenBalance({token: colToken, amount: int256(amount)});  // Unsafe cast
...
tmp[count++] = AppTokenBalance({token: WRAPPED_NATIVE, amount: int256(fee)});  // Unsafe
cast
```

4. In `_collateralOnlyBalances`:

```
balances[i] = AppTokenBalance({
    token: positions[i].addresses.collateralToken,
    amount: int256(positions[i].numbers.collateralAmount)  // Unsafe cast
});
```

While extremely large values exceeding 2^255 are unlikely in practice (this would require collateral amounts in the order of 10^59 wei for 18-decimal tokens), the lack of validation means the code does not enforce this

constraint. If GMX positions or token amounts somehow reach these values, the NAV would be catastrophically incorrect, potentially showing negative balances when they should be positive.

Disclaimer

No Guarantee of Accuracy or Completeness. Kindly note, no guarantee is being given as to the accuracy and/or completeness of any of the outputs the AuditAgent may generate, including without limitation this Report. The results set out in this Report may not be complete nor inclusive of all vulnerabilities. The AuditAgent is provided on an 'as is' basis, without warranties or conditions of any kind, either express or implied, including without limitation as to the outputs of the code scan and the security of any smart contract verified using the AuditAgent.

This Report is not a security audit. This Report has been generated automatically by AuditAgent, an AI-powered automated code scanning tool. It does not constitute, and must not be represented or relied upon as constituting, a full or comprehensive security audit conducted by Nethermind or any of its personnel. A formal security audit involves manual review by experienced human auditors and is a materially different service. If you require a full security audit, please contact Nethermind's security audit team at auditagent@nethermind.io

Use of Nethermind's name. "Nethermind" is a trade mark of Demerzel Solutions Limited. Use of this Report does not authorise you to represent that your code or smart contract has been "audited by Nethermind" or to use any similar phrase. Any such representation would be false, misleading, and an infringement of Nethermind's intellectual property rights.]
No Endorsement or Security Guarantee. Blockchain technology remains under development and is subject to unknown risks and flaws. This Report does not indicate the endorsement of any particular project or team, nor guarantee its security. Neither you nor any third party should rely on this Report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset.

Disclaimer. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this Report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.