Scanned Code Report

AUDITAGENT

Powered by NETHERMIND SECURITY

Code Info

Developer Scan

Scan ID 12	Date March 01, 2026
Organization RigoBlock	Repository v3-contracts
Branch feat/perps	Commit Hash a418940d...feeafe85

Contracts in scope

contracts/protocol/extensions/adapters/AGmxV2.sol

contracts/protocol/extensions/adapters/interfaces/IAGmxV2.sol

Code Statistics

Findings 0	Contracts Scanned 2	Lines of Code 370

Findings Summary

Total Findings

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

Code Summary

The `AGmxV2` contract is an adapter designed to facilitate interaction between a smart pool and the GMX v2 decentralized perpetuals exchange on the Arbitrum network. It is intended to be used exclusively via `delegatecall` from a parent pool contract, allowing the pool to manage leveraged trading positions on GMX.

The adapter provides a comprehensive suite of functions for position management. It enables the creation of orders to open or increase positions (`createIncreaseOrder`) and to decrease or close them using market, limit, or stop-loss orders (`createDecreaseOrder`). It also allows for the modification (`updateOrder`) and cancellation (`cancelOrder`) of pending orders. Furthermore, the contract includes functionality to claim revenue streams, such as funding fees (`claimFundingFees`) and collateral from settled positions (`claimCollateral`), ensuring these assets are returned to the pool.

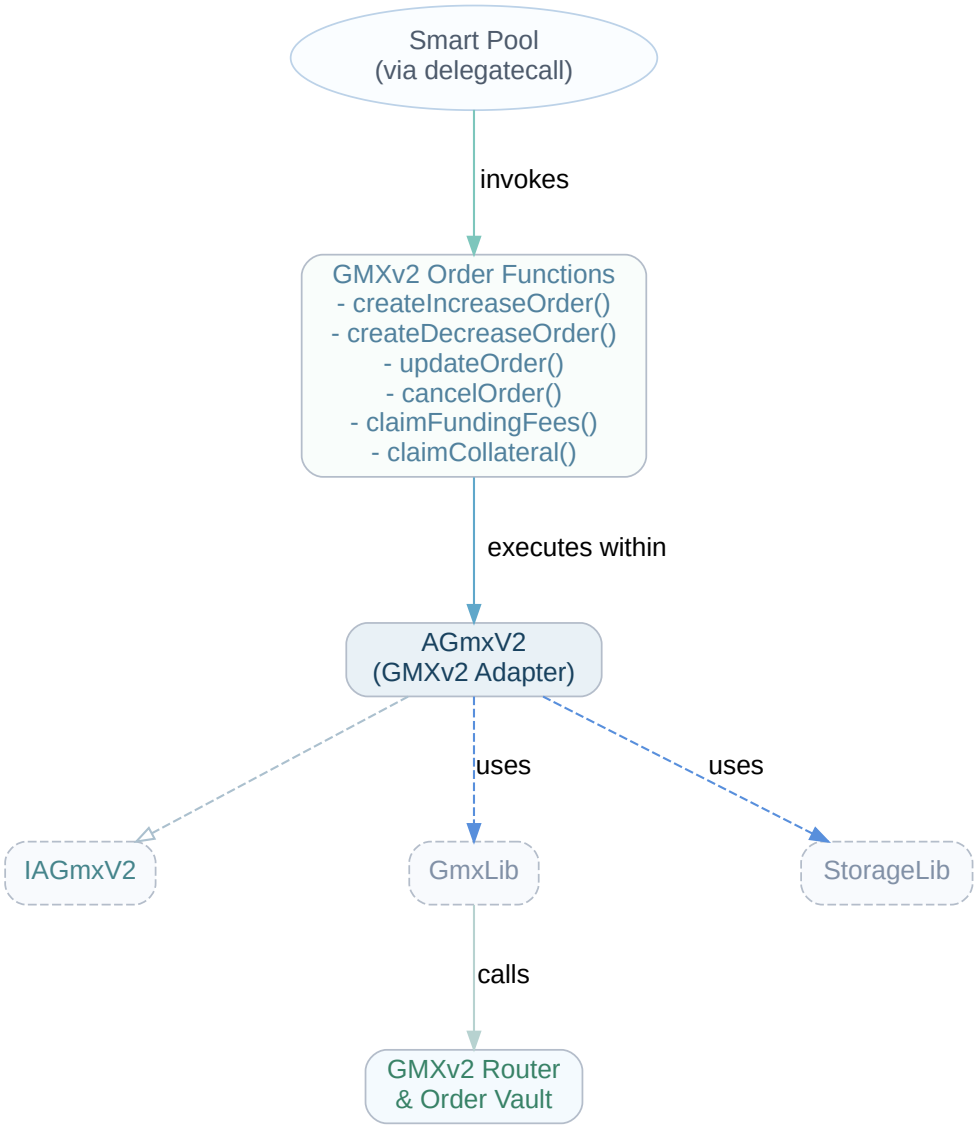
To ensure safe and efficient operation, the adapter incorporates several key mechanisms. It calculates and manages GMX execution fees on-chain, with a built-in cap to prevent excessive spending of the pool's WETH. If the pool's WETH balance is insufficient, it can automatically wrap native ETH to cover the fees. A critical feature is its token tracking system; whenever a new collateral token is used, the adapter registers it with the parent pool to ensure accurate Net Asset Value (NAV) calculations. For security and accounting simplicity, the adapter forces all funds to be received by the pool's address and ensures that when positions are closed, the returned asset is always the original collateral token.

Entry Points and Actors

The primary actor interacting with this protocol is the `Pool Manager`, who manages the assets and trading strategy of the parent smart pool.

- `createIncreaseOrder`: Allows a Pool Manager to open or increase a leveraged position on GMX by creating a market increase order.
- `createDecreaseOrder`: Allows a Pool Manager to decrease or close a position by creating a market, limit, or stop-loss decrease order.
- `updateOrder`: Allows a Pool Manager to modify the parameters of an existing, pending GMX order.
- `cancelOrder`: Allows a Pool Manager to cancel a pending GMX order, recovering the collateral and execution fees.
- `claimFundingFees`: Allows a Pool Manager to claim accumulated funding fees from GMX markets and have them deposited into the pool.
- `claimCollateral`: Allows a Pool Manager to claim collateral from positions where the negative price impact threshold was exceeded.

Code Diagram



1 of 1 Findings

contracts/protocol/extensions/adapters/AGmxV2.sol

Confusing behavior when caller provides wrong orderType in createIncreaseOrder

Best Practices

The createIncreaseOrder function accepts an IBaseOrderUtils.CreateOrderParams struct which includes an orderType field, but the adapter silently ignores this field and always forces Order.OrderType.MarketIncrease when calling GMX. This behavior is documented in the NatSpec comments, but could be confusing for integrators who populate the struct with a different order type expecting it to be used.

For comparison, createDecreaseOrder explicitly validates the order type and reverts if an unsupported type is provided:

```
solidity
require(
params.orderType == Order.OrderType.MarketDecrease ||
params.orderType == Order.OrderType.LimitDecrease ||
params.orderType == Order.OrderType.StopLossDecrease,
InvalidDecreaseOrderType()
);
```

However, createIncreaseOrder performs no such validation and instead unconditionally uses MarketIncrease. While this is safe (the caller's value is simply ignored), adding a validation check would make the behavior more explicit and fail fast if the caller expects a different order type to be used. The interface documentation states this is intentional, but explicit validation would improve clarity.

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.