GA GUARDIAN

GMX

GMX Crosschain V2.2 1

Security Assessment

July 26th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, CuriousApple, Wafflemakr, Osman Ozdemir,

Mark Jonathas, Michael Lett, Cosine

Client Firm GMX

Final Report Date July 26, 2025

Audit Summary

GMX engaged Guardian to review the security of their GMX Crosschain architecture. From the 23rd of February to the 17th of March, a team of 7 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian assigns a Confidence Ranking of 2 to the protocol. Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment. Guardian strongly advises that the protocol undergo a full follow-up audit at a finalized and fully remediated commit before any mainnet deployment. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

🔗 Blockchain network: **Arbitrum, Avalanche**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianOrg/gmx-syntheticsgmxcrosschain3-fuzz

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics/tree/main/contracts
Commit(s)	Initial commit: e809a1787b0fc531fbb764d45d81b709478c3087

Audit Summary

Delivery Date	July 26, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	9	0	0	1	0	8
High	12	0	0	0	2	10
Medium	10	0	0	2	0	8
Low	25	0	0	5	1	19
Info	0	0	0	0	0	0

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Incorrect Message Decoding Leads	Logical Error	Critical	Resolved
C-02	Front-Running lzCompose Allows Fund Theft	Validation	Critical	Resolved
C-03	Incorrect Balance Accounting	Validation	Critical	Resolved
C-04	Token Spoofing: Bridge Token X, Get Accounted For Token Y	Validation	Critical	Resolved
C-05	Order Vault Drain Via Incorrect Collateral Handling	Logical Error	Critical	Resolved
C-06	Order Vault Drain Via Incorrect Fee Deduction	Logical Error	Critical	Resolved
C-07	Missing validatePosition After Fee Deduction	Validation	Critical	Resolved
C-08	GM Tokens Sent To MultichainVault Instead Of GLV	Logical Error	Critical	Resolved
C-09	Backward Compatibility With Callback Contracts	Logical Error	Critical	Acknowledged
H-01	Pending Impact Missing In isPositionLiquidatable	Logical Error	High	Resolved
H-02	Impact Pool Cap Bypassed	Logical Error	High	Resolved
H-03	Positive Pending Impact Perturbs Negative Capping	Logical Error	High	Partially Resolved
H-04	Inability To Pay Bridge Fee For bridgeOut	Logical Error	High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
H-05	Inability To Bridge Out	Logical Error	High	Resolved
H-06	Funds Bridged To Wrong Chain bridgeOut	Validation	High	Resolved
H-07	Missing Relay Fee Payment In Case Of bridgeOut	Logical Error	High	Resolved
H-08	Transfer To EOA Instead Of Multichain Balance	Logical Error	High	Resolved
H-09	Incorrect Verification Of gasLeft	Logical Error	High	Resolved
H-10	Bypassing Validations	Validation	High	Resolved
H-11	Zero Amount Transfer Causing Reverts	Logical Error	High	Resolved
H-12	Decrease Impact Pool Cap Perturbs Price Impact	Logical Error	High	Partially Resolved
M-01	Execution Not Paid When Freezing Orders	Logical Error	Medium	Resolved
M-02	Pending Price Impact Affects pnlToPoolFactor	Logical Error	Medium	Acknowledged
M-03	Missing Max Data List Validation	Validation	Medium	Resolved
M-04	Incorrect Modifier In bridgeIn	DoS	Medium	Resolved
M-05	Missing dataList Validation In GlvHandler	Validation	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-06	Missing withOraclePricesForAtomicAction Modifier	DoS	Medium	Resolved
M-07	Bridged Amount Lower Than Expected	DoS	Medium	Resolved
M-08	Missing Refund In externalCalls For Multichain	Logical Error	Medium	Resolved
M-09	Inability To Pay Fee From Order Collateral	Logical Error	Medium	Resolved
M-10	updateOrder() Doesn't Update dataList	Logical Error	Medium	Acknowledged
L-01	Missing Max Fee Multiplier Validation	Validation	Low	Resolved
L-02	Unused _validateRange Function	Superfluous Code	Low	Resolved
L-03	Missing Params In Natspec	Documentation	Low	Resolved
L-04	Unnecessary baseSizeDeltaInTokens Declaration	Superfluous Code	Low	Resolved
L-05	Incorrect Rounding In _getProportionalImpactPendingValues	Rounding	Low	Resolved
L-06	Auto-Deleveraging (ADL) Reverts	Logical Error	Low	Acknowledged
L-07	Pending Price Updates Affect Reserve Validations	Logical Error	Low	Acknowledged
L-08	Pending Price Updates Affect Borrowing Fees	Logical Error	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	Param dataList Not Removed In Withdrawals	Logical Error	Low	Resolved
L-10	Misleading Comment For Position Fee Factor	Documentation	Low	Resolved
L-11	Multichain Routers Can Be Used From Local Chain	Logical Error	Low	Partially Resolved
L-12	Account Receives Fee Refunds	Logical Error	Low	Resolved
L-13	Missing Event In bridgeIn	Events	Low	Resolved
L-14	Multichain Balance Lost For Contracts	Validation	Low	Resolved
L-15	Same Hash If srcChainId = 0 Or = Block.chainid	Validation	Low	Acknowledged
L-16	Typo In MultichainGmRouter	Best Practices	Low	Resolved
L-17	Balance Decreased After Transfer	Best Practices	Low	Resolved
L-18	Inconsistent Use Of srcChainId For Events	Events	Low	Resolved
L-19	Token Permits Allowed For Multichain	Logical Error	Low	Resolved
L-20	Allowed _handleFeePayment For Non-Multichain	Validation	Low	Acknowledged
L-21	Incorrect Amount In emitMultichainBridgeOut	Events	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-22	Warning About Native Tokens	Documentation	Low	Resolved
L-23	Stargate Has Limited Support For Token Transfers	Logical Error	Low	Resolved
L-24	Execution Fee Params Estimated Twice	Gas Optimization	Low	Resolved
L-25	Native Tokens Not Supported In bridgeIn	Logical Error	Low	Resolved

C-01 | Incorrect Message Decoding Leads

Category	Severity	Location	Status
Logical Error	● Critical	LayerZeroProvider.sol: 68-69	Resolved

Description

GMX decodes the message received from Stargate inside `lzCompose` using:

```
(address account, address token, uint256 srcChainId) =
MultichainProviderUtils.decodeDeposit(message);
```

However, the message that Stargate sends will always be in the `OFTComposeMsgCodec` format: [stargate-v2/packages/stg-evm-v2/src/StargateBase.sol at 8ac1688ca419296c2f8c34098b3152d6c039e49c · stargate-protocol/stargate-v2]

```
composeMsg = OFTComposeMsgCodec.encode(_origin.nonce, _origin.srcEid, amountLD, _composeMsg);
```

As a result, the account, token, and `srcChainId` are incorrectly decoded, causing the `lzCompose` call to fail. Since the funds were already credited to the `LayerZeroProvider`, the user loses these funds.

The funds will instead be credited to whoever directly calls `lzCompose` with the correct token encoded in the message parameter.

Recommendation

Consider using the `OFTComposeMsgCodec` library to correctly decode the compose message and extract the parameters needed to credit the user's multichain balance.

Resolution

GMX Team: Resolved.

C-02 | Front-Running lzCompose Allows Fund Theft

Category	Severity	Location	Status
Validation	● Critical	LayerZeroProvider.sol: 61	Resolved

Description

Stargate's messaging bridge operates in a two-step process. In the first step, tokens are sent to the receiver, which in this case is the `LayerZeroProvider`, increasing the receiver's balance

Then, it is expected that `LayerZero`'s endpoint will call `lzCompose`, accounting for the balance sent in the first step. The issue here is that these two calls are not guaranteed to be atomic.

Since GMX's `LayerZeroProvider.lzCompose` is permissionless, anyone can intervene between these two steps and steal the funds deposited in `LayerZeroProvider`.

Additionally, Stargate provides a `retryReceiveToken` feature for cases where there was insufficient liquidity at the time of token delivery. When this occurs, this makes this attack even easier by publicly exposing extraction opportunity.

Recommendation

Consider:
• Restricting the `lzCompose` function so that only the `LayerZero` endpoint can call it.
• Verifying the from address and allowing it only if it originates from one of Stargate's pools.
For example:
• ReceiverStargateV2.sol - Line 92
• RangoStargateMiddleware.sol - Line 90

Resolution

GMX Team: Resolved.

C-03 | Incorrect Balance Accounting

Category	Severity	Location	Status
Validation	● Critical	LayerZeroProvider.sol: 74-75	Resolved

Description

Currently, `LayerZeroProvider` records the entire remaining balance for the account passed in `lzCompose`.

As explained in C-02, Stargate's messaging follows a two-step process:
1. Token Delivery – Tokens are sent to `LayerZeroProvider`.
2. Message Delivery – `lzCompose` recording the deposit.

Since the balance is assigned based on `lzCompose` execution, if multiple users (e.g., A, B, and C) bridge the same token with different amounts, the first user whose message gets executed in `lzCompose` receives all tokens—including those meant for others.

For example, User A could receive User B and C's funds.

Recommendation

Instead of transferring the entire balance of the `LayerZeroProvider` , use the exact bridged token amount from `OFTComposeMsgCodec` to ensure users receive only their own funds.

Example Fix:

```
uint256 amountLD = OFTComposeMsgCodec.amountLD(_message);
```

Resolution

GMX Team: Resolved.

C-04 | Token Spoofing: Bridge Token X, Get Accounted For Token Y

Category	Severity	Location	Status
Validation	● Critical	LayerZeroProvider.sol: 68-69	Resolved

Description

As explained in C-02, Stargate's messaging follows a two-step process:
1. Token Delivery – Tokens are sent to LayerZeroProvider.
2. Message Delivery – lzCompose recording the deposit.

We have already covered in another critical issues of this report that anyone can intercept these two steps and directly call the permissionless lzCompose, thereby stealing the funds.

However, even if you make lzCompose permissioned with onlyLayerZeroEndpoint and from stargate pool addresses, one critical issue still remains unresolved. Stargate has separate handlers for each token, meaning that whatever message it constructs is independent of the token itself.

composeMsg = OFTComposeMsgCodec.encode(_origin.nonce, _origin.srcEid, amountLD, _composeMsg); Relevant code reference

As a result, GMX intends to derive the token address from the _composeMsg, which the user has access to and can modify. This is where a critical issue arises. Since the _composeMsg field is a user-controlled field, one can pass any token of their choice and effectively bridge a different token using Stargate.

For example, they can pass WBTC but actually bridge USDC instead. This allows a user to steal whatever balance remains for accounting in LayerZeroProvider by frontrunning innocent's user's WBTC lzCompose.

Recommendation

Consider deriving the token address from the from parameter passed in lzCompose, instead of the msg field.

Relevant examples:
• Li.Finance contracts
• Rango Exchange contracts

Resolution

GMX Team: Resolved.

C-05 | Order Vault Drain Via Incorrect Collateral Handling

Category	Severity	Location	Status
Logical Error	● Critical	MultichainOrderRouter.sol: 151-154	Resolved

Description

With the multichain changes, GMX now allows users to create, update, or cancel orders using their balance in the Multichain vault. Users are expected to sign these changes, and Gelato is responsible for calling these functions on `MultichainOrderRouter`.

The relay fee for Gelato is paid to the fee recipient via `_handleRelayFee`. To ensure that the signing account has sufficient balance in their Multichain vault, `_handleFeePayment` is called beforehand. `_handleFeePayment` first checks if the Multichain balance is enough to cover the fee.

If it's insufficient, it attempts to recover the fee from `initialCollateralDeltaAmount` in the order and then from the user's positions by reducing their collateral. However, when transferring funds from a position, GMX incorrectly transfers the funds from the order vault instead of the market.

This allows users to pull fees from the shared order vault rather than their own positions. At the time of writing (block 22019054 on Arbitrum), the order vault balance exceeds $4M. Thus, an attacker can repeatedly call `updateOrder` with a fee, continuously draining the order vault as long as their `positionCollateralAmount` is greater than the fee.

One might argue that after the first transaction, the attacker's `positionCollateralAmount` would decrease, preventing further exploitation. However, two additional mistakes allow this attack to continue:

1. Incorrect Storage Key – The key used updates the order's key instead of the position's key, modifying unrelated storage space instead of the attacker's position.

2. Incorrect Value Update – The value set in storage is `positionCollateralAmount` instead of `positionCollateralAmount - unpaidAmount`, meaning the collateral remains unchanged.

As a result, the attacker can continue calling the function indefinitely until the order vault is empty.

Recommendation

• Ensure that the correct key and value are used when updating position collateral.
• Withdraw funds from the market instead of the order vault, as implemented here.

Resolution

GMX Team: Resolved.

C-06 | Order Vault Drain Via Incorrect Fee Deduction

Category	Severity	Location	Status
Logical Error	● Critical	MultichainOrderRouter.sol: 102	Resolved

Description

Multichain users can pay the Gelato relay fee using their Multichain balance, the pending order's initialCollateralAmount, or the position's collateralAmount. This applies only to updateOrder and cancelOrder.

However, decreaseOrders do not have an initialCollateralAmount deposited at order creation—this is only applicable to increase and swap orders.

As a result, _handleFeePayment incorrectly reduces initialCollateralAmount from the order and withdraws funds from the orderVault, effectively stealing collateral from other users and LPs. A malicious user can exploit this by:
1. Opening a position.
2. Creating a limit decrease order with initialCollateralAmount set to the total assets available in the orderVault.
3. Executing _handleFeePayment with relayParams.fee.feeAmount equal to initialCollateralAmount, which then:
• Withdraws tokens from the orderVault.
• Pays the relay fee.
• Sends the remaining amount to the attacker's Multichain balance, effectively draining the vault.

Recommendation

In _handleFeePayment, verify whether the order type is an increase or swap order—where collateral was actually deposited during order creation—before allowing the user to pay the Gelato relay fee using initialCollateralAmount.

Resolution

GMX Team: Resolved.

C-07 | Missing validatePosition After Fee Deduction

Category	Severity	Location	Status
Validation	● Critical	MultichainOrderRouter.sol: 133	Resolved

Description

Multichain users can pay the Gelato relay fee using their position's collateral if their Multichain balance or pending order's `initialCollateralAmount` is insufficient.

However, after deducting collateral from the position, there is no additional check to verify:
• Whether the position remains solvent or becomes liquidatable.
• The impact on markets, funding, and borrowing updates.

This not only allows users to arbitrarily remove collateral from their position in a single transaction but also creates bad debt for the protocol.

An attacker can front-run liquidations by updating or canceling a pending order and setting a high `feeAmount`, effectively withdrawing all collateral from their position before liquidation occurs.

Recommendation

• Do not allow paying the relay fee using the position’s collateral.
• If the protocol intends to support this feature, implement proper validation and state updates to ensure accurate collateral accounting.

Resolution

GMX Team: Resolved.

C-08 | GM Tokens Sent To MultichainVault Instead Of GLV

Category	Severity	Location	Status
Logical Error	● Critical	ExecuteGlvDepositUtils.sol: 230	Resolved

Description

During a Multichain `glvDeposit` execution, if the user deposited GM tokens, it will early return in `_processMarketDeposit` by transferring these tokens to the GLV contract.

Otherwise, if long/short tokens are deposited, an `executeDeposit` will be triggered to deposit them into the market, setting the `receiver` as the GLV contract.

However, if it's a Multichain action, the `glvDeposit.srcChainId()` is used as the `srcChainId` in the `ExecuteDepositParams`.

During the GM market deposit, specifically in `_executeDeposit`, the non-zero `srcChainId` will mistakenly mint the GM tokens to the `muiltichainVault` and record them under the GLV contract address.

Consequently, the `glvValue` calculation will be incorrect, as the GM tokens were never transferred to the GLV contract address.

This leads to insolvency when users attempt to withdraw their GLV tokens. The contract lacks sufficient `gmToken` to process withdrawals in `_processMarketWithdrawal` because a portion of its balance is tracked as a Multichain balance.

Recommendation

Set the `srcChainId` param in `ExecuteDepositParams` to zero to avoid sending the GM tokens to the `multichainVault`

Resolution

GMX Team: Resolved.

C-09 | Backward Compatibility With Callback Contracts

Category	Severity	Location	Status
Logical Error	● Critical	Global	Acknowledged

Description

The callback function signatures changed from the previous iterations (i.e. new `dataList` param was added).

This will impact with a lot of GMX integrators as they heavily rely on this callbacks, but suddenly can't receive them anymore after this update. Keep in mind that same issue applies to handlers, as the `srcChainId` was added.

Recommendation

Consider implementing a try/catch system: try the new callback function signature first and if it doesn't work try the old callback. Additionally make sure there are no issues for GMX integrations in terms of creating, updating and cancelling orders.

Resolution

GMX Team: Acknowledged.

H-01 | Pending Impact Missing In isPositionLiquidatable

Category	Severity	Location	Status
Logical Error	High	PositionUtils.sol: 318	Resolved

Description

The `isPositionLiquidatable` determines if a position can be liquidated. This check includes position PnL, price impact due to the size being closed, and fees.

However, the position might have a high negative pending price impact amount, which is not accounted for. During a liquidation, the `DecreasePositionUtils.decreasePosition()` will check if `isPositionLiquidatable` before processing collateral (where the pending price impact is applied).

Therefore, a position might be liquidatable with the pending negative price impact, but it can return false as that value is not added (or may not be liquidatable adding the positive price impact, but the check returns true).

Recommendation

Include the position's pending price impact in the `isPositionLiquidatable` check, adding it to the calculated `cache.priceImpactUsd`

Resolution

GMX Team: Resolved.

H-02 | Impact Pool Cap Bypassed

Category	Severity	Location	Status
Logical Error	High	PositionUtils.sol: 655	Resolved

Description

When executing an `increasePosition` order, the `priceImpactUsd` is calculated. In case of positive price impact, this amount is capped by the amount available in the impact pool at the time of increase. There are several issues with this behavior.

Firstly, there is a time-mismatch between when funds are paid by traders to increase the impact pool and when funds are paid out to traders from the impact pool. This issue is clear when examining a new market:

• Market A is created and has initially 0 long and 0 short open interest, the impact pool amount begins at 0
• User A opens 10 long open interest and is negatively impacted
• User A's negative impact does not increase the impact pool amount because it is stored as pending impact
• User B opens 10 short open interest and is positively impacted, but this positive impact is capped to 0 as there are no tokens in the impact pool

This time-mismatch applies to markets under normal operation as well, but is most prevalent in new markets where the impact pool starts from 0.

Secondly, there is no guarantee that the trader will actually receive the purported positive impact that is calculated based on the size of the impact pool at the time of increase. This is because the size of the impact pool can be much smaller or larger at the time of the trader's decrease order.

This presents an issue because the execution price on increase is computed with the assumption that the exact price impact amount will be ultimately realized by the trader. This behavior can be misleading to users and will result in positions which do not receive execution prices in line with the `acceptablePrice` specified.

Recommendation

To solve both issues, consider taking the following measures:

• Apply delta to the position impact pool for the positive or negative impact which is applied on increase
• Track the summation of the `pendingPriceImpact` across all positions in a market, with a `totalPendingPriceImpactAmount` variable
• Add the `totalPendingPriceImpactAmount` value in USD to the pool value in the `getPoolValueInfo` function to offset the difference in the `positionImpactPool`.
• Do not apply delta to the position impact pool for the pending impact on decrease, instead only apply the delta for the price impact associated with the decrease order that is being currently executed

Resolution

GMX Team: Resolved.

H-03 | Positive Pending Impact Perturbs Negative Capping

Category	Severity	Location	Status
Logical Error	High	DecreasePositionCollateralUtils.sol: 145	Partially Resolved

Description

When a user has positive pending price impact this is combined with the price impact of the decrease order being executed to compute the `totalImpactUsd`.

This is done before the negative price impact is capped, therefore during times of volatility when the negative price impact cap would be in use users with positive pending price impact will experience loss without receiving their price impact rebates.

Recommendation

When the `proportionalImpactPendingUsd` is positive, then add it to the final `collateralCache.totalImpactUsd` after the negative price impact cap is applied.

Resolution

GMX Team: Partially Resolved.

H-04 | Inability To Pay Bridge Fee For bridgeOut

Category	Severity	Location	Status
Logical Error	High	LayerZeroProvider.sol: 85	Resolved

Description

The `LayerZeroProvider.bridgeOut` requires a native fee value sent along `stargate.send()`. This native value is required to pay for the bridging fee.

However, the `bridgeOut` function is not payable and there is no logic to withdraw ETH from the Multichain user balance.

Recommendation

Consider implementing some logic for the user to pay for the bridging fee, withdrawing WETH from multichain balance to unwrap it.

Resolution

GMX Team: Resolved.

H-05 | Inability To Bridge Out

Category	Severity	Location	Status
Logical Error	High	LayerZeroProvider.sol: 107	Resolved

Description

When bridging out tokens using the `LayerZeroProvider`, the user withdraws these tokens from the multichain balance. Additionally, a `nativeFee` value needs to be sent to the stargate pool to pay for the bridge fee.

However, the function mistakenly withdraws `valueToSend` instead of the `amount` of tokens. This `valueToSend` is the native fee calculated to pay for the bridge fee and not `amount + bridging fee` as stated in the comment.

Therefore, when `stargate.send()` is executed, the `safeTransferFrom` will revert as the tokens are not available in the router balance.

Recommendation

Withdraw the `amount` instead of `valueToSend` from multichain balance

Resolution

GMX Team: Resolved.

H-06 | Funds Bridged To Wrong Chain bridgeOut

Category	Severity	Location	Status
Validation	High	MultichainTransferRouter.sol: 39	Resolved

Description

In the `getBridgeOutStructHash` the `data` value is not included in the resulting hash and therefore cannot be validated. This param is the `dstEid` used when sending a message through `LayerZero`.

Therefore, a malicious user with the user's signature can send a relay transaction through Gelato with a different `data` param.

Validations will succeed as the signature does not include this param, and funds will be bridged to a different chain from what user requested. Additionally, the `provider` param is also not included in the struct hash.

Even though there is a validation that checks if this address is whitelisted in GMX, there could be different providers and the transaction might be executed with the wrong address.

Recommendation

Include the `data` and `provider` parameters in the `getBridgeOutStructHash`

Resolution

GMX Team: Resolved.

H-07 | Missing Relay Fee Payment In Case Of bridgeOut

Category	Severity	Location	Status
Logical Error	High	MultichainTransferRouter.sol: 39	Resolved

Description

The MultichainTransferRouter allows users to bridge funds in or out of their multichain balance. These are Gelato relay enabled functions to allow gasless transactions.

This relay will execute callWithSyncFeeV2, the function in charge to call the GMX router contracts with a specific fee amount, token and collector address encoded in the calldata.

The bridgeOut function contains the onlyGelatoRelay modifier. In order to fully enable the gasless feature, the fee in feeToken needs to be paid to the feeCollector before the execution finishes.

However, there is no logic to pay for this relay fee, different from the other routers. Consequently, the bridgeOut function can't be gasless.

Recommendation

Implement the _handleRelay function to pay the Gelato relay fee to the feeCollector.

Resolution

GMX Team: Resolved.

H-08 | Transfer To EOA Instead Of Multichain Balance

Category	Severity	Location	Status
Logical Error	High	Global	Resolved

Description

In multiple places throughout the codebase, tokens that should have been included in the multichain balance are instead transferred directly to the user's EOA address. This issue occurs when transferring the residual fee in the `BaseGelatoRelayRouter`.

• In `_cancelOrder`, the residual fee receiver is the account itself, and the balance is transferred to the EOA instead of the `multichainVault`.
• In `_updateOrder`, similarly, the residual fee receiver is the account itself when `increaseExecutionFee` is false.

Additionally, the same issue occurs when multichain orders fail during the execution process. While successful executions are handled as multichain, failed executions and refunds are processed as non-multichain actions. This issue occurs in the following cases:
• `_handleDepositError` during multichain GM deposits.
• `_handleGlvDepositError` during multichain GLV deposits.
• `_handleWithdrawalError` during multichain GM withdrawals.
• `_handleGlvWithdrawalError` during multichain GLV withdrawals.
• `_handleShiftError` during multichain GM shifts.
• `_handleOrderError` when any multichain orders fail during the execution phase.
• `_handleSwapError` when decreasing positions, even though the outer position decrease doesn't fail, but only the inner swap fails.

Lastly, the issue occurs when users cancel their multichain order using `MultichainOrderRouter.cancelOrder`. The cancellation is processed as a non-multichain order because `OrderUtils.cancelOrder` doesn't check `srcChainId` when determining the `cancellationReceiver` here. However, the `executionFeeReceiver` is determined based on `srcChainId` here.

Recommendation

Use the `srcChainId` when handling execution errors and during cancellations to determine the correct destination for transferring the tokens. If tokens are transferred to `MultichainVault`, `recordTransferIn` must be called post that to record the balance for the account.

Resolution

GMX Team: Resolved.

H-09 | Incorrect Verification Of gasLeft

Category	Severity	Location	Status
Logical Error	High	Global	Resolved

Description

As you can in following code, GMX has added the createEventData call for all callbacks (after executions) starting from multichain push.

However, since GMX performs the gasleft() / 64 * 63 > callbackGasLimit check before calling createEventData, the receiver may end up with a lower callbackGasLimit than what is specified in the try/catch block.

This happens because createEventData consumes some of the available gas (gasleft()), effectively forcing the catch block to trigger.

If the callback does not receive the required gas, it can lead to critical issues in downstream integrations. These integrations may fail to validate or update their state, causing a desynchronization with GMX.

Recommendation

Consider validating gas left : validateGasLeftForCallback(order.callbackGasLimit()) , after OrderEventUtils.createEventData(order)

```
EventUtils.EventLogData memory orderData = OrderEventUtils.createEventData(order);
validateGasLeftForCallback(order.callbackGasLimit());
```

Resolution

GMX Team: Resolved.

H-10 | Bypassing Validations

Category	Severity	Location	Status
Validation	High	MultichainGlvRouter.sol: 98-99	Resolved

Description

MultichainGmRouter: _createShift
MultichainGlvRouter: _createGlvWithdrawal

Both functions allow users to bypass key validations in their respective handlers by leveraging Multichain routers.

Since they directly use internal utility libraries instead of handlers, they skip critical checks such as:
1. Feature availability
2. Global reentrancy modifier
3. Data list length validation

Recommendation

Consider using dedicated handlers instead of directly using internal library functions.

Resolution

GMX Team: Resolved.

H-11 | Zero Amount Transfer Causing Reverts

Category	Severity	Location	Status
Logical Error	High	Global	Resolved

Description

With the Multichain update, whenever the receiver is a multichain vault, `recordTransferIn` must be called afterward to account for the transfer.

However, since `recordTransferIn` reverts on zero amount transfers, GMX must always ensure amount > 0 before calling it.

There are instances throughout the codebase where this check is missing, causing a revert if the amount is zero, leading to a DoS risk for the related actions.

Affected Areas:
1. Multichain Decrease Orders
• DecreaseOrderUtils.sol, Lines 90-107
2. Multichain Withdrawals
• ExecuteWithdrawalUtils.sol, Lines 324-328
3. Residual Fee Transfers
• MultichainRouter.sol, Lines 86-87

There are other occurrences, such as in swaps, but these are less likely to trigger the issue due to existing validations (e.g., `minOut`).

Recommendation

• If reverting on zero amount transfers is unnecessary, consider returning early instead of reverting to avoid potential DoS risks.
• If reverting is essential, ensure an `amount > 0` check is present everywhere `recordTransferIn` is called.

Resolution

GMX Team: Resolved.

H-12 | Decrease Impact Pool Cap Perturbs Price Impact

Category	Severity	Location	Status
Logical Error	High	DecreasePositionCollateralUtils.sol: 89	Partially Resolved

Description

When a position is decreased the positive price impact for the decrease is capped at the amount in the price impact pool. Later on the total price impact (decrease + pending) is capped again at the amount in the price impact pool.

Capping the total price impact is necessary but capping the decrease price impact can lead to loss for the user. For example:
• The amount in the price impact pool is 200
• A user closes a position with a pending price impact of -300
• The user receives a price impact of 300 for the decrease
• Therefore the total price impact the user should receive is 300 - 300 = 0
• But in reality, the user receives a negative price impact as the price impact from the decrease was capped at the amount in the price impact pool (200)
• Therefore the user received 200 - 300 = -100 price impact

As we can see this cap is unnecessary before calculating the total price impact, as the total price impact would not even have touched the price impact pool. This could therefore punish users who balance the market and therefore have a negative effect on the health of the system.

Recommendation

Remove the capping of positive price impact for the execution of the decrease order and rely only on the capping of `totalImpactUsd`. The effective price impact of the decrease order used for execution price should then be calculated as:

```
decreaseOrderImpact = totalCappedImpactUsd - pendingImpactUsd
```

This ensures the correct validation of the user's acceptable price.

Resolution

GMX Team: Partially Resolved.

M-01 | Execution Not Paid When Freezing Orders

Category	Severity	Location	Status
Logical Error	Medium	OrderUtils.sol: 310	Resolved

Description

During `OrderUtils.freezeOrder`, the `executionFee` is set to 0. This is the same value passed to `GasUtils.payExecutionFee`. which early returns if its 0.

Therefore, the order keeper does not get paid the execution fee, and the user does not receive the refund.

This opens the possibility of gas griefing the keeper, where a malicious user can update their order, force it to fail and become frozen; keeper receives no compensation.

Recommendation

Cache the current order execution fee before setting it to 0, and pass that value to `payExecutionFee` instead.

Resolution

GMX Team: Resolved.

M-02 | Pending Price Impact Affects pnlToPoolFactor

Category	Severity	Location	Status
Logical Error	Medium	Global	Acknowledged

Description

Previously, whenever a position was opened, the price impact was realized instantly. And the pool as a whole used to realize the PnL depending on the price impact. For example, if the price impact was positive, the trader's overall PnL in `getPoolValueInfo` would decrease, reducing the value of the market token. If the price impact was negative, the trader's overall PnL would increase, distributing that profit to LPs over time. This was enforced using the corresponding change in `openInterestInTokens` while positions were opened.

However, now that traders receive a `sizeDeltaInTokens` in correspondence with the market price, `openInterestInTokens * indexTokenPrice` is exactly equal to `openInterestSize` for that particular trader, and the price impact incurred is stored as pending. This pending price impact, depending on its sign, is nothing but profit or loss for the pool as far as LPs are concerned. Since it is not considered in functions like `getPnl()`, this causes several issues around the `pnlToPoolFactor` which affects several areas of the exchange.

1. `isPnlFactorExceeded`
This function may report false positives or false negatives, leading to unintended consequences: In cases where a user has large positive pending price impact ADLs may be allowed when they shouldn't be or disallowed when they should be. Considering both their PnL and pending price impact, their position should now be auto-deleveraged. But since `getPnl()` does not consider the pending price impact, the ADL wouldn't go through.

This can impact the deposit and withdrawal behavior as well, because `isPnlFactorExceeded` does not verify whether pending price impacts on positions push the pool beyond the allowed PnL factor. As a result, actions like withdrawals or deposits may still be processed even if the `WithdrawalPnLFactor` or `DepositPnLFactor` has already been exceeded.

2. `getPositionPnlUsd`
Previously, price impact was already factored into PnL, ensuring a trader's PnL-to-pool ratio stayed within the specified limits. Now, traders would no longer be guaranteed to be capped at a predefined max PnL-to-pool factor percentage.

3. `getPnlToPoolFactor`
The value could be either lower or higher than reality, creating potential problems for integrators. Integrators rely on `getPnlToPoolFactor` returning what they consider the correct value. However, with the v2.2 change incorporating pending price impact, this creates a discrepancy between the state assumed by integrators and the actual state in GMX, potentially leading to costly mismatches.

Recommendation

Consider factoring in pending price impacts within `openInterestInTokens`, ensuring that `getPnl()` accurately reflects the current state of the pool.

Resolution

GMX Team: Acknowledged.

M-03 | Missing Max Data List Validation

Category	Severity	Location	Status
Validation	Medium	WithdrawalHandler.sol: 38	Resolved

Description

The `executeAtomicWithdrawal` function does not perform the `validateDataListLength` validation on the `dataList` provided on the withdrawal object.

This may cause issues for integrations which do not expect the `dataList` to be above the max length in the callback, or experience unexpected reverts due to OOG of performing memory operations with a sufficiently large `dataList`.

Recommendation

Perform the `validateDataListLength` validation in the `executeAtomicWithdrawal` function.

Resolution

GMX Team: Resolved.

M-04 | Incorrect Modifier In bridgeIn

Category	Severity	Location	Status
DoS	Medium	MultichainTransferRouter.sol: 29	Resolved

Description

The `bridgeIn` function has the `onlyGelatoRelay` modifier; however, this function is intended to be called by users via a multicall to record their balances.

Recommendation

Remove the `onlyGelatoRelay` and relevant validations for `bridgeIn`

Resolution

GMX Team: Resolved.

M-05 | Missing dataList Validation In GlvHandler

Category	Severity	Location	Status
Validation	Medium	GlvHandler.sol: 135-143	Resolved

Description

The `createGlvWithdrawal` function in the `GlvHandler` contract does not validate the `dataList` length, unlike the deposit function.

Recommendation

Call `validateDataListLength` during the `createGlvWithdrawal` function as well.

Resolution

GMX Team: Resolved.

M-06 | Missing withOraclePricesForAtomicAction Modifier

Category	Severity	Location	Status
DoS	Medium	Global	Resolved

Description

The relay fee can be paid using a token other than wnt by swapping it in the `_swapFeeTokens` function, which requires token prices to be set.

However, actions requiring a fee token swap in `MultichainGlvRouter`, `MultichainGmRouter`, and `MultichainTransferRouter` lack the `withOraclePricesForAtomicAction` modifier. As a result, token prices are not set for the swap, causing fee payments to fail.

Recommendation

Add the `withOraclePricesForAtomicAction` modifier and set prices for atomic swaps.

Resolution

GMX Team: Resolved.

M-07 | Bridged Amount Lower Than Expected

Category	Severity	Location	Status
DoS	Medium	LayerZeroProvider.sol: 86	Resolved

Description

Stargate Protocol is used to bridge funds in and out to Arbitrum/Avalanche. When bridging out, the `prepareSend` function is in charge of calculating the bridge fee and the amounts sent and received in Local Decimals as `OFTReceipt` {`amountSentLD`, `amountReceivedLD`}.

However, the amount requested by the user might not be equal to the amountSentLD, for the following reasons:

• Token Transfer Precision
where `amountSentLD` is rounded down for tokens decimals not equal to 6 (shared decimals)
• Path's Limits

Consequently, this issue causes impacts in both `ERC20` and ETH bridging:

• `ERC20`: Using `approve()` may cause a DoS for some `ERC20` tokens if stargate does not transfer all amount, and allowance is non-zero after bridging.
• ETH: Stargate will treat the precision lost in the amount as excess bridge fee and refund this to the account directly.

Recommendation

Instead of using the `amount` function param, consider using the `OFTReceipt.amountSentLD` for the approval and the `SendParam.amountLD` struct.

Resolution

GMX Team: Resolved.

M-08 | Missing Refund In externalCalls For Multichain

Category	Severity	Location	Status
Logical Error	Medium	BaseGelatoRelayRouter.sol: 274	Resolved

Description

Multichain users have the option to use the `externalHandler` to perform actions outside of GMX. This can be utilized to use Uniswap swap feature and get a better output amount.

However, the `_handleRelayFee` only handles scenarios where 100% of the `feeToken` is converted to WNT (i.e. `swapExactIn`).

In case other actions are used (i.e. `swapExactOut`), the handler will refund both the `feeToken` and `wnt` to the GMX router.

After the external calls execution, only the `wnt` amount is recorded in the multichain balance as residual fee.

Even if the external call has a `refundToken` and `refundReceiver` set for the `feeToken`, it can only be sent to the user, but never recorded in the user's multichain balance.

Recommendation

After the external call is executed, consider verifying the contract's balance for the `feeToken`, send it to the `multichainVault` and record the transfer for the user's balance.

Resolution

GMX Team: Resolved.

M-09 | Inability To Pay Fee From Order Collateral

Category	Severity	Location	Status
Logical Error	Medium	MultichainOrderRouter.sol: 95	Resolved

Description

Multichain users are allowed to pay for fees using their Multichain balance, order collateral or position collateral.

In case of pending swap orders or limit orders without position, it should only allow payment from collateral in the order, as positions do not exist for these orders.

However, the function reverts with `UnableToPayOrderFee` if `relayParams.fee.feeToken = position.collateralToken()`.

This check is performed before collateral is deducted from the order. As `position.collateralToken()` is `address(0)` for the orders mentioned above, it will always revert.

Recommendation

Consider validating the `feeToken` against the `order.initialCollateralToken()` instead.

Resolution

GMX Team: Resolved.

M-10 | updateOrder() Doesn’t Update dataList

Category	Severity	Location	Status
Logical Error	Medium	OrderHandler.sol: 93-94	Acknowledged

Description

Currently, `updateOrder` doesn't allow updating the `dataList`, a feature GMX intends to allow.

Recommendation

Consider allowing updation of `dataList` in `updateOrder` with the validation of `validateDataListLength`

Resolution

GMX Team: Acknowledged.

L-01 | Missing Max Fee Multiplier Validation

Category	Severity	Location	Status
Validation	Low	ConfigUtils.sol	Resolved

Description

In the `validateRang` function in the `ConfigUtils` file the `MAX_EXECUTION_FEE_MULTIPLIER_FACTOR` validation has been errantly removed.

Recommendation

Add the `MAX_EXECUTION_FEE_MULTIPLIER_FACTOR` validation to the end of the `validateRange` function in the `ConfigUtils` file.

Resolution

GMX Team: Resolved.

L-02 | Unused _validateRange Function

Category	Severity	Location	Status
Superfluous Code	Low	Config.sol	Resolved

Description

In the `Config` contract the `_validateRange` function is no longer used and can be removed.

Recommendation

Remove the `_validateRange` function from the `Config` contract.

Resolution

GMX Team: Resolved.

L-03 | Missing Params In Natspec

Category	Severity	Location	Status
Documentation	Low	Global	Resolved

Description

With the new addition of `dataList` and `srcChainId` multiple functions are missing the `NatSpec` for these new params.

Recommendation

Consider reviewing all functions with these new params and add the corresponding `NatSpec`.

Resolution

GMX Team: Resolved.

L-04 | Unnecessary baseSizeDeltaInTokens Declaration

Category	Severity	Location	Status
Superfluous Code	Low	PositionUtils.sol: 694	Resolved

Description

In the `getExecutionPriceForIncrease` function on line 694 the `cache.baseSizeDeltaInTokens` value is unnecessarily referenced, with no use.

Recommendation

Remove this reference to the `cache.baseSizeDeltaInTokens` value.

Resolution

GMX Team: Resolved.

L-05 | Incorrect Rounding In _getProportionalImpactPendingValues

Category	Severity	Location	Status
Rounding	Low	DecreasePositionCollateralUtils.sol: 740	Resolved

Description

In the `_getProportionalImpactPendingValues` function the `proportionalPendingImpactAmount` is always rounded towards zero.

For cases where positions have a positive pending impact amount this correctly rounds down the user's positive impact.

However for cases where the user experiences negative impact, the protocol should round in the direction of a larger magnitude of negative impact rather than a smaller magnitude.

Recommendation

Consider using an overloaded `mulDiv` function call which accepts a `roundUpMagnitude` boolean to round the magnitude up only in the case where users experience a negative price impact.

Resolution

GMX Team: Resolved.

L-06 | Auto-Deleveraging (ADL) Reverts

Category	Severity	Location	Status
Logical Error	Low	AdlHandler.sol: 147-148	Acknowledged

Description

In the current version of GMX, while executing ADL, GMX first executes the order and then checks whether the pnlToPoolFactor has improved. However, due to how pnlToPoolFactor is validated, this process can lead to unnecessary reverts for ADLs.

pnlToPoolFactor = PnL / poolValueUSD (excluding PnL and the position impact pool)

Example Scenario:
Initial state of the pool:
• PnL = 30
• Pool Value = 100
• pnlToPoolFactor = 30 / 100 = 0.3
After Auto-Deleveraging (ADL) a Position with 5 PnL:
• New PnL = 25
• New Pool Value = 95
• nextPnlToPoolFactor = 25 / 95 = 0.26

Since 0.3 > 0.26, the condition passes, and ADL proceeds as expected. Now, consider a scenario where closing a position benefits from a positive price impact. In this case, the denominator (pool value) in nextPnlToPoolFactor decreases further, inflating the ratio and potentially triggering an unnecessary revert

Example with Positive Price Impact:
• Trader receives a positive price impact of 15
• Adjusted Pool Value = 80
• nextPnlToPoolFactor = 25 / 85 = 0.3125

Since 0.3125 > 0.3, the revert condition triggers, blocking the ADL. These cases are rare as the decrease in PnL from large profitable positions will in the majority of cases overshadow the decrease in pool value resulting from positive impact.

However with the addition of the pending price impact it is more likely that this edge case arises because any previously unrealized positive price impact now gets realized during the decrease.

Recommendation

Be aware of the increased likelihood of this edge case resulting in blocked ADL orders. Otherwise consider accounting the positive price impacts in the validations done post ADL order execution to allow the ADLing of positions which meet this edge case.

Resolution

GMX Team: Acknowledged.

L-07 | Pending Price Updates Affect Reserve Validations

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

Previously, whenever a position was opened, the price impact was realized instantly. As a result, traders either received a higher or lower number of index tokens based on the current market price, with a corresponding change in `openInterestInTokens`.

However, now that traders receive an execution price equal to the market price, `openInterestInTokens` is either inflated or deflated compared to its actual intended value. This is later adjusted through the pending price impact when the position is closed.

However, other actions that depend on `openInterestInTokens` before the user closes their position would be affected in the interim. For example, methods using `getReservedUsd`, which directly depends on `getOpenInterestInTokens`, or vaults/strategies built on top of GMX that reference `getOpenInterestInTokens`.

Now, depending on the pending impact across all positions, the `getReservedUsd` function would return:
• Inflated `reservedUsd` if the sum of pending price impacts is negative
• Deflated `reservedUsd` if the sum of pending price impacts is positive

This would result in the reserve validations with `validateReserve` and `validateOpenInterestReserve` either allowing actions when they technically should not be allowed or not allowing actions when they technically should be allowed.

Recommendation

Be aware of this change in behavior and consider accounting for the total pending impact in the reserve validation if parity with the previous version is desired for these validations.

Resolution

GMX Team: Acknowledged.

L-08 | Pending Price Updates Affect Borrowing Fees

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

Previously, whenever a position was opened, the price impact was realized instantly. As a result, traders either received a higher or lower number of index tokens based on the current market price, with a corresponding change in `openInterestInTokens`.

However, now that traders receive an execution price equal to the market price, `openInterestInTokens` is either inflated or deflated compared to its actual intended value. This is later adjusted through the pending price impact when the position is closed.

However, other actions that depend on `openInterestInTokens` before the user closes their position would be affected in the interim. For example, methods using `getReservedUsd`, which directly depends on `getOpenInterestInTokens`, or vaults/strategies built on top of GMX that reference `getOpenInterestInTokens`.

As a result, traders may end up paying higher or lower borrowing fees than they should. Since borrowing fees are directly proportional to the ratio of reserves to pool value, incorrect reserve calculations can cause deviations in borrowing fees.

Recommendation

Be aware of this change in behavior and consider accounting for the total pending impact in the borrowing calculations if parity with the previous version is desired for borrowing rates.

Resolution

GMX Team: Acknowledged.

L-09 | Param dataList Not Removed In Withdrawals

Category	Severity	Location	Status
Logical Error	Low	WithdrawalStoreUtils.sol: 202-203	Resolved

Description

The `WithdrawalStoreUtils` contains logic to set and get the new `dataList` param, but there is not a logic to remove it from storage.

Although there is no clear impact besides gas related issues, this param should be removed to avoid future conflicts, just like its removed in all other actions.

Recommendation

Add the `removeBytes32Array` in `WithdrawalStoreUtils.remove()` to remove `dataList` from storage.

Resolution

GMX Team: Resolved.

L-10 | Misleading Comment For Position Fee Factor

Category	Severity	Location	Status
Documentation	Low	PositionPricingUtils.sol: 497	Resolved

Description

When increasing or decreasing a position there are cases where `balanceWasImproved = true` but the `priceImpactUsd` is negative.

This happens when the position incurs both positive and negative price impact values and the negative price impact factor is larger than the positive impact factor.

The `balanceWasImproved` is used in `getPositionFeesAfterReferral()` when the `positionFeeFactor` is calculated. According to the comment `in this case the fee factor for the negative price impact would be charged`.

However, if `balanceWasImproved` is true, then the POSITIVE position fee factor is used, so the comment above is incorrect. The same issue applies to `getSwapFees` .

Recommendation

Update the comment about the `positionFeeFactor`:

`in this case the fee factor for the positive price impact would be charged` for the case when `priceImpactUsd` is negative and `balanceWasImproved`.

Resolution

GMX Team: Resolved.

L-11 | Multichain Routers Can Be Used From Local Chain

Category	Severity	Location	Status
Logical Error	Low	Global	Partially Resolved

Description

The Multichain routers are supposed to be used in a Multichain context but can be used from the local chain. This might lead to unexpected behaviour or could introduce bugs in the future.

Recommendation

Add a check that the given `srcChainId` is not zero or the local chain id.

Resolution

GMX Team: Partially Resolved.

L-12 | Account Receives Fee Refunds

Category	Severity	Location	Status
Logical Error	Low	LayerZeroProvider.sol: 114	Resolved

Description

During the `stargate.send()` call, the `account` is set as the refund address. In case there is any surplus sent for the bridge fee, this refund address will receive native tokens back.

The current implementation sets `refundAddress` as the `account`, but as this is a multichain transaction, all funds should be credited to the multichain balance instead.

The fee quote and actual message sending is done in the same transaction, so it's not likely that there will be any refunds. Nevertheless, if there is any refund in the future, the `account` address should not receive it.

Recommendation

The `stargate.send` returns a `MessagingReceipt`. This should be used to verify if there was any refund.

Consider setting the `MultichainVault` as the `refundReceiver` and trigger a `MultichainUtils.recordTransferIn` if there is a difference from the amount paid and the `nativeFee` sent.

Resolution

GMX Team: Resolved.

L-13 | Missing Event In bridgeIn

Category	Severity	Location	Status
Events	Low	MultichainTransferRouter.sol: 36	Resolved

Description

The `bridgeIn` function in the `MultichainTransferRouter` contract calls `MultichainUtils.recordTransferIn` instead of `MultichainUtils.recordBridgeIn`. Therefore the `emitMultichainBridgeIn` event is not emitted.

Recommendation

Use `MultichainUtils.recordBridgeIn` instead or emit a `bridgeIn` event in the `bridgeIn` function.

Resolution

GMX Team: Resolved.

L-14 | Multichain Balance Lost For Contracts

Category	Severity	Location	Status
Validation	Low	MultichainTransferRouter.sol: 39-63	Resolved

Description

Smart contracts / smart accounts are not able to sign transactions and therefore can't use the Multichain functions. This means when a smart contract receives funds in the Multichain vault the funds are lost forever.

Not every end user might be aware of this fact and know how the Multichain feature works on such a technical level. This can therefore likely lead to user mistakes with big financial consequences.

For example:
• A user is active on multiple chains and also Arbitrum and Base (like most traders in the web3 space)
• The user has liquidity on Base and wants to use it in GMX on Arbitrum and decides to use the new Multichain feature to do so
• The user opens a position and decides to close it after a while
• As the user also has a smart wallet on Arbitrum which he usually uses for GMX trades he decides to save himself the bridging fees and set the smart wallet as the receiver of the decrease order
• The funds are lost forever as the smart wallet can't interact with the Multichain functions

Recommendation

Add a checks to ensure that the order receiver is not a smart contract.

Resolution

GMX Team: Resolved.

L-15 | Same Hash If srcChainId = 0 Or = Block.chainid

Category	Severity	Location	Status
Validation	Low	BaseGelatoRelayRouter.sol: 329	Acknowledged

Description

The `BaseGelatoRelayRouter._validateCall` function treats a multichain transaction with `srcChainId = 0` equal to a multichain transaction with `srcChainId = block.chainid`.

This is a dangerous pattern as the system checks if an order is local or multichain based on a `srcChainId = 0` check and could therefore lead to exploits in future versions of the protocol.

Recommendation

Remove this line in `_validateCall`: `uint256 _srcChainId = srcChainId = 0 block.chainid : srcChainId;` and revert instead if the given `srcChainId` is the current chain

Resolution

GMX Team: Acknowledged.

L-16 | Typo In MultichainGmRouter

Category	Severity	Location	Status
Best Practices	Low	MultichainGmRouter.sol: 73	Resolved

Description

"...will be recorder as relay fee" comment in line 73 of `MultichainGmRouter` should be recorded.

Recommendation

Fix the typo.

Resolution

GMX Team: Resolved.

L-17 | Balance Decreased After Transfer

Category	Severity	Location	Status
Best Practices	Low	MultichainUtils.sol: 94-95	Resolved

Description

In MultichainsUtils.transferOut, tokens are transferred first, and then the balance is decremented, which is contrary to best practices.

```
multichainVault.transferOut(token, receiver, amount);
dataStore.decrementUint(Keys.multichainBalanceKey(account, token), amount);
```

Recommendation

Consider following the CEI pattern and transferring tokens only after decrementing the balance.

Resolution

GMX Team: Resolved.

L-18 | Inconsistent Use Of srcChainId For Events

Category	Severity	Location	Status
Events	Low	Global	Resolved

Description

The `MultichainUtils.recordTransferIn` is used to record funds sent to the `multichainVault`, and add them to the user's tracked balance. The `srcChainId` param is only used to emit an event with the id of the action's source chain.

However, there are multiple parts of the code where this `srcChainId` is hardcoded to 0 and some others where the action's `srcChainId` is used (i.e. used when executing `glvDeposit` but not used when executing a normal deposit).

Although there is no major impact in the contract, events will be emitted with the wrong param for multichain users.

Additionally, according to the changelog doc, in the future `the balance for contracts will be separated by chainId`. Therefore, this param can actually have an impact if its value is set to 0 or the `srcChainId`.

Recommendation

Send the action's `srcChainId` to the `recordTransferIn` so the event is emitted with the correct value. Alternatively, if the idea of hardcoding the `srcChainId` to 0 was to point out where the call originated from (normal action or Gelato multichain), be sure to be consistent with the value used during executions.

Resolution

GMX Team: Resolved.

L-19 | Token Permits Allowed For Multichain

Category	Severity	Location	Status
Logical Error	Low	MultichainRouter.sol: 53	Resolved

Description

Multichain actions will utilize user's Multichain balance collateral. The `MultichainRouter` overrides the `_sendTokens` so that it will use `MultichainUtils.transferOut` instead of `router.pluginTransfer`.

Consequently, there is no need for token permits, specially because user is not suppose to have funds in the protocol's chain. Additionally, this permit will be left unused.

Recommendation

For multichain actions, verify that `tokenPermits.length = 0`.

Resolution

GMX Team: Resolved.

L-20 | Allowed _handleFeePayment For Non-Multichain

Category	Severity	Location	Status
Validation	Low	MultichainOrderRouter.sol: 74	Acknowledged

Description

The `_handleFeePayment` feature allows Multichain users to use position's and order's collateral to pay for relay and execution fee.

This is not a feature enabled for non-multichain routers. However, there is no validation for `srcChainId`, so users can create a gasless transaction with `srcChainId = 0`, and freely move collateral from order to multichain balance.

Recommendation

If this is intended behavior, consider documenting it for users. Otherwise, validate if `srcChainId` is non zero and equal to the `order.srcChainId`.

Resolution

GMX Team: Acknowledged.

L-21 | Incorrect Amount In emitMultichainBridgeOut

Category	Severity	Location	Status
Events	Low	MultichainTransferRouter.sol: 60	Resolved

Description

The `bridgeOut` functionality will emit an event `emitMultichainBridgeOut` after the bridge transaction is made using Stargate.

However, there are cases where the amount requested is not the amount actually sent, due to rounding or path limits, so the event will be emitted with the incorrect amount.

Recommendation

Return the amount actually sent through Stargate in the `LayerZeroProvider.bridgeOut()`, and use this amount in the event emitted.

Resolution

GMX Team: Resolved.

L-22 | Warning About Native Tokens

Category	Severity	Location	Status
Documentation	Low	LayerZeroProvider.sol	Resolved

Description

The `LayerZeroProvider` and Multichain contracts are not designed to hold native assets. However, Stargate supports native token transfers, allowing users to initiate transfers from any source chain to a destination chain.

As a result, while Stargate will accept the user's native tokens on the source chain, the `LayerZeroProvider` will be unable to receive these tokens on the destination chain, leading to a loss of funds for the user.

Recommendation

Document this behavior and warn users against transferring native tokens via Stargate.

Resolution

GMX Team: Resolved.

L-23 | Stargate Has Limited Support For Token Transfers

Category	Severity	Location	Status
Logical Error	Low	Lack Of Present Code	Resolved

Description

Stargate has limited support for tokens that it can transfer. Through withdrawals and swaps users can obtain tokens that cannot be bridged back to their original chains.

This will force users to open additional positions in order to swap their funds to the proper assets, and can cause them to face price impact, slippage, and fees.

Recommendation

Add a function that will allow users to create swaps with their multichain vault balances. Additionally, document to users that they should verify the tokens they receive can be bridged back to their desired blockchain.

Resolution

GMX Team: Resolved.

L-24 | Execution Fee Params Estimated Twice

Category	Severity	Location	Status
Gas Optimization	Low	OrderUtils.sol: 179	Resolved

Description

During order creation, the `estimatedGasLimit` and `oraclePriceCount` is calculated once, stored in the cache struct, but then `validateAndCapExecutionFee` does not use these values, as it calculates them again.

Recommendation

Use the cache values from previous calculation in `validateAndCapExecutionFee`.

Resolution

GMX Team: Resolved.

L-25 | Native Tokens Not Supported In bridgeIn

Category	Severity	Location	Status
Logical Error	Low	LayerZeroProvider.sol: 75	Resolved

Description

The `bridgeOut` flow supports bridging native tokens using the `StargatePoolNative` contract.

However, there is no support to bridge these token in, as it will require a `receive` function in the `LayerZeroProvider` contract, and an alternate flow to wrap ETH and record it as WETH balance in `multichainVault`.

Recommendation

Consider adding a receive function and handle special case for `token = address(0)`, wrapping the ETH received and recording the WETH in `multichainVault`.

Resolution

GMX Team: Resolved.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits