GA GUARDIAN

GMX

GMX Crosschain V2.2 2

Security Assessment

July 26th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, CuriousApple, Wafflemakr,

Cosine, Mark Jonathas, Osman Ozdemir

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 24th of February to the 5th of May, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of non-critical issues detected and code changes following the main review, Guardian assigns a Confidence Ranking of 3 to the protocol. Guardian advises the protocol to address issues thoroughly and consider a targeted follow-up audit depending on code changes. 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: d7a23d1a6a940be929b429e9f5f18629da6c3b03

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	2	0	0	0	0	2
High	5	0	0	0	0	5
Medium	13	0	0	4	0	9
Low	25	0	0	6	0	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	Missing Validations	Logical Error	Critical	Resolved
C-02	Inconsistent Accounting	Logical Error	Critical	Resolved
H-01	Transfer To EOA Instead Of Multichain Balance	Logical Error	High	Resolved
H-02	Position Validated With Atomic Oracles	Validation	High	Resolved
H-03	Missing Expo In messageHash	Validation	High	Resolved
H-04	Price Impact Not Guaranteed	Logical Error	High	Resolved
H-05	Uncapped Positive Impact In Liquidation Check	Validation	High	Resolved
M-01	Missing validateSequencerUp	Validation	Medium	Resolved
M-02	Incorrect Balance Check In bridgeOut	Logical Error	Medium	Resolved
M-03	Nonce Always Increased	DoS	Medium	Resolved
M-04	Missing validateSequencerUp In Impact Pool	Validation	Medium	Resolved
M-05	Collateral Mismatch	Logical Error	Medium	Resolved
M-06	Early Validation Allows Inconsistent Market State	Validation	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-07	Stale Funding State Used	Validation	Medium	Resolved
M-08	Interface Changes	Warning	Medium	Acknowledged
M-09	Missing Disable integrationId Feature	Logical Error	Medium	Acknowledged
M-10	Integration Id Not Validated For Gasless Routers	Validation	Medium	Resolved
M-11	Price Deviation In Impact Pool Withdraw	Oracle	Medium	Acknowledged
M-12	Imbalance In Impact Pool From ADL & Liquidations	Logical Error	Medium	Acknowledged
M-13	validatePosition Might DoS Valid Actions	Logical Error	Medium	Resolved
L-01	Untrusted srcChainId	Events	Low	Resolved
L-02	Bridged Funds Can Be Lost Or Trapped	Informational	Low	Resolved
L-03	Misleading Comment For Bridge Out Fee	Documentation	Low	Resolved
L-04	Bridge Out Same Chain Requires Two Transfers	Gas Optimization	Low	Resolved
L-05	Unnecessary Approval For WNT When Bridging	Logical Error	Low	Resolved
L-06	transferOut Perturbs Multichain Accounting	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-07	Incorrect recordTransferIn Usage	Logical Error	Low	Resolved
L-08	Modifier Execution Order	Logical Error	Low	Resolved
L-09	Incorrect srcChainId Used	Events	Low	Acknowledged
L-10	Precision Loss For Token Withdrawal	Rounding	Low	Resolved
L-11	Last Market Not Validated	Validation	Low	Acknowledged
L-12	Unused Cache Struct Param	Superfluous Code	Low	Resolved
L-13	Inconsistent Validation	Validation	Low	Resolved
L-14	Missing Feature Flag Validation	Validation	Low	Resolved
L-15	Liquidation Optimization createLiquidation	Gas Optimization	Low	Resolved
L-16	Frontrunning Risk	Warning	Low	Acknowledged
L-17	Missing Safeguards	Validation	Low	Resolved
L-18	Old Orders Can Overwrite Newer Ones	Validation	Low	Resolved
L-19	Integration Ids Can't Be Set With Multichain	Logical Error	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-20	Validate Receiver In batchClaimAffiliateRewards	Validation	Low	Resolved
L-21	Impact Pool Withdrawals May Constantly Revert	Informational	Low	Acknowledged
L-22	Risk Free Trade By Causing Reverts	Validation	Low	Resolved
L-23	Incorrect Gas Tracking For Relay	Logical Error	Low	Resolved
L-24	Frontrunning withdrawFromPositionImpactPool	Warning	Low	Acknowledged
L-25	Unnecessary srcChainId In bridgeIn	Events	Low	Resolved

C-01 | Missing Validations

Category	Severity	Location	Status
Logical Error	● Critical	MultichainOrderRouterUtils.sol: 37-156	Resolved

Description

The `transferFeeFromOrderOrPosition` allows users to withdraw collateral from positions. The `validatePosition` intends to verify that positions are still within size and collateral range, and not in a liquidatable state. If the check passes, collateral is withdrawn from the market.

Nonetheless, critical state changes are not enforced to guarantee the security of the protocol:

• `applyDeltaToCollateralSum` as funds are withdrawn from market. If omitted, it can lead to a complete DoS of the market as the `validateMarketTokenBalance` safety check could revert.

• `updateFundingAndBorrowingState` as fees applied to the position are incorrectly calculated, leading to invalid liquidatable status.

• `incrementClaimableFundingAmount` and `handleReferral` to remain consistent with the decrease position flow.

Recommendation

Consider removing the feature to allow withdrawing collateral from a position. If this is a non-negotiable feature, ensure all state changes are applied before validating the position, or use the `decreasePosition` flow instead.

Resolution

GMX Team: Resolved.

C-02 | Inconsistent Accounting

Category	Severity	Location	Status
Logical Error	● Critical		Resolved

Description

GMX maintains two fields for tracking a position's pending price impact: `pendingImpactAmount` (in index tokens) and `pendingImpactUsd` (in USD). However, only `pendingImpactAmount` is correctly prorated during position decreases.

In the decrease process, the USD impact is re-calculated as:

`proportionalPendingImpactUsd = proportionalPendingImpactAmount × currentIndexPrice`

instead of proportionally reducing the stored `pendingImpactUsd`.

This leads to discrepancies when the index price changes between position increase and decrease, causing:
• `pendingImpactUsd` to exceed its initial value,
• `pendingImpactUsd` to become negative despite a positive remaining token amount,
• Inaccurate liquidation checks, which rely on `pendingImpactUsd`, potentially liquidating healthy positions or causing reverts.

Example:
1. A user opens a position with index token price at $1:
◦ `pendingImpactAmount` = 1000 tokens
◦ `pendingImpactUsd` = $1000
2. The price increases to $2, and the user closes 75% of the position:
◦ `proportionalPendingImpactAmount` = 750 tokens
◦ `proportionalPendingImpactUsd` = 750 × $2 = $1500
3. Updated values:
◦ `pendingImpactAmount` = 250 tokens
◦ `pendingImpactUsd` = $1000 − $1500 = -$500

Recommendation

Ensure both token- and USD-denominated impacts are prorated from their own stored totals, rather than recalculating one from the current price. Or consider removing the explicit tracking for `pendingPriceImpactUsd` itself and default to using `pendingPriceImpact * indexTokenPrice`

Resolution

GMX Team: Resolved.

H-01 | Transfer To EOA Instead Of Multichain Balance

Category	Severity	Location	Status
Logical Error	High	Global	Resolved

Description

Tokens that should have been included in the multichain balance are instead transferred directly to the user's EOA address.

This 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.

- `_handleDepositError`
- `_handleGlvDepositError`
- `_handleWithdrawalError`
- `_handleGlvWithdrawalError`
- `_handleShiftError`

Recommendation

Use the `srcChainId` when handling execution errors 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-02 | Position Validated With Atomic Oracles

Category	Severity	Location	Status
Validation	High	MultichainOrderRouterUtils.sol: 128	Resolved

Description

The validatePosition function is now called in the transferFeeFromOrderOrPosition to check if the position is liquidatable and if it meets the minimum position size and collateral in usd.

However the _handleRelayBeforeActionForOrders uses a an atomic oracle (chainlink price feed) used to fetch the market prices, which lacks behind the real price by the given deviation value.

There are multiple impacts that arise from this issue:

1. Min position collateral may be breached

2. If deviation is against the user, users might not be able to take the collateral or remaining might insufficient (liquidatable) so transaction reverts.

3. If deviation is with the user needs, users will be able to withdraw more collateral from position than they should be allowed to, as the deviation gives more room to reach the liquidatable status.

Additionally, if the deviation is 0.5% or more, and the user manages to frontrun the oracle update or the keeper liquidation, there is a risk of introducing bad debt to the protocol.

Recommendation

Our primary recommendation is to reconsider if this feature is truly essential and remove it otherwise. Alternatively, consider adding a extra buffer to market prices, so it validatePosition reverts earlier.

Resolution

GMX Team: Resolved.

H-03 | Missing Expo In messageHash

Category	Severity	Location	Status
Validation	High	EdgeDataStreamVerifier.sol: 99-100	Resolved

Description

The `messageHash` used for verifying signed price data does not include the expo field. As a result, when using the Chaos Labs Edge Data Stream provider, a malicious caller can supply an arbitrary expo value, thereby manipulating the effective price, since the price is derived as:

```
price = value * 10^abs(expo)
```

Chaos Labs' publicly available price schema does not expose bid, ask, or clarify whether expo is included in the signed payload.

Furthermore, the public key required to verify Chaos' signatures is only shared during integration, so we were unable to independently verify whether expo is part of the signed message or not.

Depending on Chaos' signing scheme:

• If expo is included in the Chaos signature, signature verification will always revert on GMX.

• If expo is not included, it will allow price manipulation.

Recommendation

• If Chaos includes `expo` in their signed payload, update the GMX `messageHash` to include `expo`.

• If Chaos does not include `expo`, enforce `expo` per feed as a configuration value in GMX's `DataStore`.

• We strongly recommend testing signature verification using a real Chaos API response. If Chaos can provide a sample signed payload and the corresponding public key, we'd be happy to verify the correctness of the logic.

Resolution

GMX Team: Resolved.

H-04 | Price Impact Not Guaranteed

Category	Severity	Location	Status
Logical Error	High	Global	Resolved

Description

H-02 of the previous review is partially resolved. Even though the capping with impact pool amount during increase was removed from the code, issues related to time mismatch still persists. Since the delta amount is only applied during decrease orders, users with positive impact still miss out of their rebates.

For example:
• 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, which is also stored as pending impact.
• User C opens 10 short open interest and is negatively impacted.
• User C's negative impact also does not increase the impact pool amount because it is stored as pending impact.
• User B closes his position and is positively impacted. But this impact is still capped at 0 even though both of user B's actions had positive impacts.
• User A and C close their positions with negative pending impact and impact pool balance is increased.

In this scenario, User B would get their pending positive impact if they close the position after User A and C, but they won't get if they close it before. The ordering of executions significantly impacts users' price impact rebates and users has no power on this. It will depend on the executor/keeper's actions and blockchain's nature.

Recommendation

Consider implementing recommended changes in the previous H-02 and applying delta at the time of increase.

Resolution

GMX Team: Resolved.

H-05 | Uncapped Positive Impact In Liquidation Check

Category	Severity	Location	Status
Validation	High	PositionUtils.sol: 350-357	Resolved

Description

In the `isPositionLiquidatable` function the price impact for closing the position is summed up with the pending price impact and the resulting total impact is capped at 0 (can only be negative).

The price impact for closing the position is not capped based on the max positive price impact factor before adding the pending price impact (like during the actual decrease order).

This will therefore influence the validation and can result in positions appearing healthy which are actually liquidatable.

For example:
• A position is liquidatable (in reality)
• Its positive price impact for closing the position would be capped at $500 due to the max positive price impact factor
• The pending price impact of the position is -$700
• The `isPositionLiquidatable` function calculates a price impact of $800 for closing the position
• The price impact is summed up with the pending price impact: $800 + (-$700) = 100 and therefore it is capped at 0
• If the positive price impact would have been capped at 500 (as in the actual decrease order), the result would have been: 500 + (-$700) = -$200 instead and this difference could have flagged the position as liquidatable instead of healthy

Recommendation

Cap the positive price impact for closing the position based on the max positive price impact factor before summing it up with the pending price impact in the `isPositionLiquidatable` function.

Resolution

GMX Team: Resolved.

M-01 | Missing validateSequencerUp

Category	Severity	Location	Status
Validation	Medium	MultichainOrderRouterUtils.sol: 37-156	Resolved

Description

The `transferFeeFromOrderOrPosition` function uses an atomic oracle to validate position (enabled market, collateral token, liquidatable status), whenever collateral is used to pay relay fees.

However, there is a missing validation of the `oracle.validateSequencerUp()` to make sure that the arbitrum sequencer is up and running.

In case the sequencer is down this `validatePosition` calculations could be drastically off. Although the sequencer check is present when using atomic swaps, not all multichain transactions will involve a GMX v2 swap.

Recommendation

Execute `oracle.validateSequencerUp()` before validating the position.

Resolution

GMX Team: Resolved.

M-02 | Incorrect Balance Check In bridgeOut

Category	Severity	Location	Status
Logical Error	Medium	LayerZeroProvider.sol: 177	Resolved

Description

During the `bridgeOut` flow, the bridging fee of `wnt` is transferred from the user's multichain balance, then unwrapped to the native token to pay the fee.

A balance check ensures the native transfer was successful. If it fails, the function is intended to return early without bridging and transfers tokens back to the user's multichain balance.

However, it is not possible for `wntBalanceAfter` to be greater than `wntBalanceBefore`. It will either decrease (if successful) or stay the same (if unsuccessful). As a result, the `if (wntBalanceAfter > wntBalanceBefore)` block won't be executed even if the native transfer fails.

The function will not return early as intended and will continue executing, which can lead to one of two impacts:

1. Bridging will be performed using the contract's native balance: The contract will hold a native token balance due to incoming bridging actions, as there's a time gap between the token transfer and the `lzCompose` call. The `stargate.send` will be executed using these incoming tokens, which will cause the subsequent `lzCompose` to fail, resulting in a loss of funds for the user attempting to bridge in.
2. Bridging will fail due to insufficient balance: If there is insufficient native token balance, the `stargate.send` call will revert. This will expose the user's signature, which remains valid because the nonce is not incremented on revert. As a result, the same signature can be reused later by anyone.

Recommendation

Check if the `wntBalanceAfter` is equal to `wntBalanceBefore` to determine whether the native transfer has failed. Alternatively, compare the difference in the `wnt` balance against the requested withdrawal amount to ensure they are equal.

Resolution

GMX Team: Resolved.

M-03 | Nonce Always Increased

Category	Severity	Location	Status
DoS	Medium	MultichainSubaccountRouter.sol: 176	Resolved

Description

The `_handleSubaccountAction` function increases the nonce even if no `subaccountApproval` signature was given.

This could lead to DoS when multiple sub accounts interact with one main account and the main account tries to approve a sub account via a signature.

Recommendation

Only increase the nonce when a signature was validated.

Resolution

GMX Team: Resolved.

M-04 | Missing validateSequencerUp In Impact Pool

Category	Severity	Location	Status
Validation	Medium	MarketPositionImpactPoolUtils.sol: 36	Resolved

Description

The `withdrawFromPositionImpactPool` function is executed with atomic oracle prices using `executeWithOraclePrices`.

Therefore, sequencer uptime should be validated before using `getMarketPrices` to correctly calculate the pool value info.

Recommendation

Make sure on chain prices are correct using `validateSequencerUp` check.

Resolution

GMX Team: Resolved.

M-05 | Collateral Mismatch

Category	Severity	Location	Status
Logical Error	Medium	MultichainOrderRouterUtils.sol: 37-38	Resolved

Description

This function derives the position key using `order.initialCollateralToken()`. However, GMX allows the user to swap from initial to actual collateral during execution of the order.

As a result:

• The derived key might not match (causing revert), or

• It might unintentionally reference and deduct another position.

Recommendation

Consider using actual execution collateral to construct the correct key for the position. (if there is a swap path, last `tokenOut`)

Resolution

GMX Team: Resolved.

M-06 | Early Validation Allows Inconsistent Market State

Category	Severity	Location	Status
Validation	Medium	MarketPositionImpactPoolUtils.sol: 106-107	Resolved

Description

The `validateMarketTokenBalance` check is currently performed before the transfer of long and short tokens within `withdrawFromPositionImpactPool`.

This ordering is incorrect, as it can allow `withdrawFromPositionImpactPool` to pass validation even when it shouldn't, potentially leaving the market in an inconsistent state.

Recommendation

Move the `validateMarketTokenBalance` call to occur after the `transferOut` operations.

Resolution

GMX Team: Resolved.

M-07 | Stale Funding State Used

Category	Severity	Location	Status
Validation	Medium	MarketPositionImpactPoolUtils.sol: 46-47	Resolved

Description

The `withdrawFromPositionImpactPool` function includes validations based on `validateMarketTokenBalance`.

• `validateMarketTokenBalance` considers claimable funding fees.

However, since GMX does not call `updateFundingAndBorrowingState` prior to these validations, they operate on stale state, potentially causing the function to:

• Revert when it shouldn't, or

• Pass when it should revert.

Recommendation

Call `updateFundingAndBorrowingState` after `distributePositionImpactPool` and before any validations that depend on funding or borrowing data.

Resolution

GMX Team: Resolved.

M-08 | Interface Changes

Category	Severity	Location	Status
Warning	Medium	Global	Acknowledged

Description

Several integrations of GMX have recently faced issues due to changes in reader contract interfaces — particularly when function signatures were updated without backward compatibility. Similar concerns now arise with the v2.2 changes, which modify both reader functions and state-changing flows, potentially causing integration breakage unless proactively addressed.

Reader Contract Changes Examples:
Some commonly used reader functions have had their signatures or return types modified:
1. `isPositionLiquidatable`
• `forLiquidation` is added as function input.
• Integrations relying on this may now receive a function not found error.
Reference →

2. `getAccountOrders`
• Return type has changed from `Order.Props[]` memory to `ReaderUtils.OrderInfo[]` memory, which now includes the order key.
• This may cause deserialization or parsing issues for integrations expecting the previous format.
Reference →

3. `getExecutionPrice` (via `ReaderPricingUtils`)
• The return struct has changed.
• Integrations parsing pricing data or simulating execution logic may break silently or return incorrect values.
Reference →

State-Changing Flow Changes (Previously Reported):
Additionally, state-changing functions across the board — including `createDeposit`, `createWithdrawal`, `createOrder`, and associated callbacks — are impacted by:
• The introduction of the `DataStore.Data[]` datalist parameter.
• Addition of `sourceChainId` as a required param.
• Callback behavior modifications.

These changes are non-trivial and will require coordinated updates by integrators.

Recommendation

We recommend that the GMX team proactively notify all integrators and provide:
• A migration guide or changelog.
• Sample diffs or updated SDK adapters.
• Adequate lead time before deploying changes to production networks.

Resolution

GMX Team: Acknowledged.

M-09 | Missing Disable integrationId Feature

Category	Severity	Location	Status
Logical Error	Medium	Global	Acknowledged

Description

Users can add or remove sub accounts, granting some privileges to execute orders on behalf of the account with some limitations.

Additionally, users can set an `integrationId` for each sub account so that the protocol can intervene if there are some compromised sub account keys, by disabling all sub account for a certain integration.

However, there is no function that allows the protocol to disable an integration id.

Recommendation

Create a `disableIntegrationId` function protected by protocol admin to allow disabling integration ids.

Resolution

GMX Team: Acknowledged.

M-10 | Integration Id Not Validated For Gasless Routers

Category	Severity	Location	Status
Validation	Medium	SubaccountRouterUtils.sol: 24	Resolved

Description

Subaccounts using the `SubaccountRouter` may create, update or cancel orders on behalf of the account.

Each of these actions trigger the `_handleSubaccountAction` which validates subaccount feature, integration id, and handles subaccount action.

Although a similar process is done by all actions in `SubaccountGelatoRelayRouter` and `MultichainSubaccountRouter`, these routers use the `SubaccountRouterUtils.handleSubaccountAction` which does not contain the integration id validation.

Recommendation

Consider adding the integration id validation in `SubaccountRouterUtils.handleSubaccountAction` : `SubaccountUtils.validateIntegrationId(dataStore, account, subaccount);`

Resolution

GMX Team: Resolved.

M-11 | Price Deviation In Impact Pool Withdraw

Category	Severity	Location	Status
Oracle	Medium	MarketPositionImpactPoolUtils.sol: 29-126	Acknowledged

Description

The `withdrawFromPositionImpactPool` allows GMX to withdraw from the price impact pool. To do so it removes the amount from the position impact pool and it tries to remove an equal amount of long and short tokens from the pool itself. Usually this should cancel each other out and the LPs do not gain or lose funds because of this action.

But as Chainlink oracle prices are used to do so the price deviation could lead to accidentally removing too much or too less long/short tokens in comparison to the removed impact pool amount.

As both the index price and the long/short token prices could be off, the stepwise jump in the LP share price could be up to 1% of the withdrawn impact pool amount (as a lot of price feeds on Arbitrum and Avalanche have a deviation value of 0.5%).

As the function is executed with a timelock controller, LPs are able to know when a `withdrawFromPositionImpactPool` can be expected and might therefore be able to sandwich the call and gain from the action or avoid losses.

Recommendation

Consider using recent prices instead of Chainlink prices in the `withdrawFromPositionImpactPool` function.

Resolution

GMX Team: Acknowledged.

M-12 | Imbalance In Impact Pool From ADL & Liquidations

Category	Severity	Location	Status
Logical Error	Medium	DecreasePositionCollateralUtils.sol: processCollateral()	Acknowledged

Description

On a decrease position, the user will have to pay for any funding owed, negative PnL, fees, and then any negative price impact that the position has incurred. A scenario can arise that the user is unable to pay the entirety of their debt to the protocol.

In a normal decrease order this will lead to a revert, however, when this occurs through ADL or liquidations an event is emitted and the function will return early.

Since the impact pool amount is the last debt to be repaid, this leads to an imbalance in the impact pool amount, and no funds will be added to pay the traders who have received positive price impact.

Recommendation

Similarly to when funding fees can not be paid, an event should be emitted for the impact pool as well.

This event should signify that the insurance fund needs to add assets to the pool amount to cover price impact.

Additionally, an admin privileged function should be added in order to update the position impact pool amount.

Resolution

GMX Team: Acknowledged.

M-13 | validatePosition Might DoS Valid Actions

Category	Severity	Location	Status
Logical Error	Medium	MultichainOrderRouterUtils.sol: 138	Resolved

Description

To fix a previous issue, a call to `validatePosition` was added after subtracting collateral from the position during the relay fee payment flow.

This call is made with both `shouldValidateMinPositionSize` and `shouldValidateMinCollateralUsd` set to true.

When decreasing positions, the same check is performed, but with these values set to false. As a result, it is possible for a position with a dust-sized amount to remain after a decrease, such as after a liquidation.

Normally, this wouldn't be a problem if the user intends to increase the position and add more collateral later, since the next validation would occur only after the increase order is executed.

However, this becomes problematic in a multichain context, as the check occurs well before the position update, as part of `_handleRelayBeforeActionForOrders`.

A multichain `updateOrder` or `cancelOrder` call may fail due to previously remaining dust positions, even if the actions themselves are valid.

Recommendation

Consider setting `shouldValidateMinCollateralUsd` to true and `shouldValidateMinPositionSize` to false during this validation in `transferFeeFromOrderOrPosition`.

Resolution

GMX Team: Resolved.

L-01 | Untrusted srcChainId

Category	Severity	Location	Status
Events	Low	LayerZeroProvider.sol: 76	Resolved

Description

In the `lzCompose` function the `srcChainId` is decoded from the `composeMessage`, however the `composeMessage` is controllable by the user.

Therefore the `srcChainId` can be incongruent with the chain the user actually bridged from. This may cause issues for consumers of the `MultichainBridgeIn` and `MultichainTransferIn` events.

Recommendation

Instead, the trusted `srcEid` included in the message in the `lzCompose` call should be used to infer the source chain. If desired the `srcEid` can be mapped back to a `srcChainId`.

Resolution

GMX Team: Resolved.

L-02 | Bridged Funds Can Be Lost Or Trapped

Category	Severity	Location	Status
Informational	Low	LayerZeroProvider.sol: 63	Resolved

Description

When bridging funds using Stargate, users must carefully construct the transaction to ensure funds are delivered correctly and the `lzCompose` function is executed as intended.

Several parameters are critical:

• `composeMsg`: If omitted or malformed, `lzCompose` will either never be called or will revert on every attempt, resulting in funds being irreversibly trapped.

• `account`: If incorrectly set, funds may be credited to an unintended user in the `MultichainVault`.

• `srcChainId`: Must match the actual source chain to ensure event emissions on the destination chain are valid.

• `receiver`: Must be set to the `LayerZeroProvider` contact address, as it is the recipient for funds and where `lzCompose` is invoked by the `LayerZero` `EndpointV2`.

The current system lacks a mechanism for recovering funds when a malformed `composeMsg` causes repeated reverts, leading to permanent fund lockup in the `LayerZeroProvider`.

Recommendation

• Enforce validation and thorough documentation of all Stargate bridging parameters in the GMX UI to minimize user error.

• Consider introducing a trusted admin function to allow the protocol to recover or redirect funds that are otherwise trapped due to malformed messages or misconfigurations.

Resolution

GMX Team: Resolved.

L-03 | Misleading Comment For Bridge Out Fee

Category	Severity	Location	Status
Documentation	Low	LayerZeroProvider.sol: 187	Resolved

Description

The `LayerZeroProvider.bridgeOut` withdraws `wnt` tokens from the multichain vault in order to pay the bridge fee stored in `cache.valueToSend`.

However, this value can also include the amount being bridged when interacting with the `StargatePoolNative` contract.

Therefore, the `bridge out fee` comment on `cache.valueToSend` is misleading and may confuse the reader.

Recommendation

Add a note to the comment, explaining that this value may also include `amountSentLD`, not just the bridge fee.

Resolution

GMX Team: Resolved.

L-04 | Bridge Out Same Chain Requires Two Transfers

Category	Severity	Location	Status
Gas Optimization	Low	MultichainTransferRouter.sol: 81	Resolved

Description

When withdrawing tokens from Multichain vault using `MultichainTransferRouter.transferOut`, the same-chain withdrawal path will first call `MultichainUtils.transferOut` to transfer tokens from the `MultichainVault` to the router, and then a second transfer from the router to the account, spending more gas than needed.

Recommendation

During `MultichainUtils.transferOut`, set the receiver as the `account` to avoid a second token transfer

Resolution

GMX Team: Resolved.

L-05 | Unnecessary Approval For WNT When Bridging

Category	Severity	Location	Status
Logical Error	Low	LayerZeroProvider.sol: 151	Resolved

Description

When bridging out funds using `StargatePoolNative`, both the bridge fee and `amountSentLD` are sent as native tokens in `stargate.send{ value: cache.valueToSend }`. Therefore, there is no need to approve `Stargate` for WNT tokens.

Recommendation

Avoid approving WNT tokens to Stargate pool when `stargate.token() = address(0x0)`.

Resolution

GMX Team: Resolved.

L-06 | transferOut Perturbs Multichain Accounting

Category	Severity	Location	Status
Logical Error	Low	StrictBank.sol: 89	Resolved

Description

In the `StrictBank` contract an overloaded version of the `_recordTransferIn` function has been added which accepts a distinct amount value to transfer in.

This is to allow individual cross-chain token transfers to be recorded individually upon `lzCompose` execution.

However the `_afterTransferOut` function sets the recorded balance to the current `StrictBank` contract balance.

When there are funds which have been bridged but not yet recorded with the corresponding `lzCompsose` call, this will perturb the accounting and validation performed in the `_recordTransferIn` function.

This is because the `nextBalance` will be incremented past the recorded `tokenBalances` entry causing reverts upon `lzCompose` execution and trapping bridged tokens.

Recommendation

Use an overridden implementation of the `transferOut` function for the `MultichainVault` contract, whereby the specific amount is decremented from the `tokenBalances` entry, rather than setting the entry to the current balance.

Resolution

GMX Team: Resolved.

L-07 | Incorrect recordTransferIn Usage

Category	Severity	Location	Status
Logical Error	Low	MultichainUtils.sol: 74	Resolved

Description

In the `MultichainUtils.recordTransferIn` function which is used after order execution to record outputs, the `multichainVault.recordTransferIn(token)` function is used.

This function does not include the specific token amount being recorded in and therefore will overwrite the `tokenBalances` entry to the current balance of the `MultichainVault` contract.

This will perturb the `tokenBalances` accounting for cross-chain bridges and will result in in-flight bridges which have not yet had their `lzCompose` call recorded being stuck due to reverts after the order execution action takes place and invokes `multichainVault.recordTransferIn(token)`.

Recommendation

Use the `multichainVault.recordTransferIn(token, amount)` overloaded function which includes the amount being recorded in the `MultichainUtils.recordTransferIn` function.

Resolution

GMX Team: Resolved.

L-08 | Modifier Execution Order

Category	Severity	Location	Status
Logical Error	Low	MultichainOrderRouter.sol: 42	Resolved

Description

In Solidity, when a function has multiple modifiers, they are executed in the order they appear in the function declaration (from left to right).

Although most functions declare `withRelay` and then `nonReentrant`, it's not the same case for the `MultichainOrderRouter.batch`.

Although there is no clear attack path, as permits are now blocked for multichain transactions, it's safer avoid potential issues.

Recommendation

In order to remain consistent, update the order of modifiers so `nonReentrant` is declared first, and then the `withRelay` modifier after it.

Resolution

GMX Team: Resolved.

L-09 | Incorrect srcChainId Used

Category	Severity	Location	Status
Events	Low	Global	Acknowledged

Description

According to the documentation:

`srcChainId has been added to support multichain, e.g. if an order was signed from Base then broadcast on Arbitrum, the srcChainId would be the chainId of Base. For same chain actions, i.e. a non multichain action, the srcChainId would be zero`

However, this `srcChainId` value does not seem consistent in the codebase with the previous statement, as there are cases where the order is multichain, but a hardcoded value of 0 is used instead of the actual `srcChainId`, or same chain orders using a non zero value.

Some examples:
• `MultichainTransferRouter.transferOut`: calls `MultichainUtils.transferOut` with the `block.chainId` instead of 0.

• `DecreaseOrderUtils.processOrder`: call `MultichainUtils.recordTransferIn` with the actual `srcChainId` value, but `DecreaseOrderUtils._handleSwapError` uses a hardcoded value of 0 for `recordTransferIn`.

• `OrderUtils.cancelOrder`: even if it's a multichain order, it uses a value of 0 when returning funds to user

• `GasUtils.payExecutionFee`: id is always 0, no matter what `srChainId` param is.

Recommendation

Verify all instances where `srcChainId` is used, and make sure the correct value is set to signal a multichain order.

Resolution

GMX Team: Acknowledged.

L-10 | Precision Loss For Token Withdrawal

Category	Severity	Location	Status
Rounding	Low	MarketPositionImpactPoolUtils.sol: 84	Resolved

Description

The current calculation for `longTokenWithdrawalAmount` and `shortTokenWithdrawalAmount` will divide before multiplying.

Recommendation

Although precision loss is just one or two wei, it's better to leave the division by two for the denominator param.

Resolution

GMX Team: Resolved.

L-11 | Last Market Not Validated

Category	Severity	Location	Status
Validation	Low	BaseGelatoRelayRouter.sol: 293-295	Acknowledged

Description

During a gasless transaction, users are able to swap tokens to WNT in order to pay the relay fee. After the swap, a market validation is execute for the `feeSwapPath`.

However, the loop iterates from index 0 to the `feeSwapPath.length - 1`, meaning that the last market in the path is not validated.

This is different from what `ExecuteDepositUtils.swap` function validates after performing a swap, where all the `swapPathMarkets` are validated.

Recommendation

Do not subtract the 1 from the path length, so it will also validate the last market address.

Resolution

GMX Team: Acknowledged.

L-12 | Unused Cache Struct Param

Category	Severity	Location	Status
Superfluous Code	Low	MultichainOrderRouterUtils.sol: 30C17-30C45	Resolved

Description

The `TransferFeeFromOrderOrPositionCache` is used as a struct to store temporal data in the `transferFeeFromOrderOrPosition` function.

However, the `initialCollateralDeltaAmount` param is never used, as the function creates a temporal variables instead of using the cache struct.

Recommendation

Consider using the cache struct param or remove this from the struct if not in use.

Resolution

GMX Team: Resolved.

L-13 | Inconsistent Validation

Category	Severity	Location	Status
Validation	Low	SubaccountRouter.sol	Resolved

Description

`setIntegrationId` correctly checks sub account existence via `validateSubaccount()`. However, other functions of sub account router like `setSubaccountExpiresAt()` and `setSubaccountAutoTopUpAmount()` do not.

Recommendation

Add `validateSubaccount` to all sub account-related setters for consistency.

Resolution

GMX Team: Resolved.

L-14 | Missing Feature Flag Validation

Category	Severity	Location	Status
Validation	Low	ReferralUtils.sol: 116-117	Resolved

Description

The batchClaimAffiliateRewards function includes a feature flag check:

```
FeatureUtils.validateFeature(dataStore,
Keys.claimAffiliateRewardsFeatureDisabledKey(address(this)));
```

However, the standalone claimAffiliateReward function does not include this validation, even though it is marked as public.

Currently, this isn't an issue since claimAffiliateReward isn't being used anywhere. But if there are plans to expose or use this function in the future, the lack of feature flag validation could lead to inconsistent behavior or unintentional access.

Recommendation

If claimAffiliateReward is intended for future use, consider adding the same validateFeature call to it for consistency and control. Alternatively, make it internal or private if it's not meant to be directly used.

Resolution

GMX Team: Resolved.

L-15 | Liquidation Optimization createLiquidation

Category	Severity	Location	Status
Gas Optimization	Low	LiquidationUtils.sol: 74-75	Resolved

Description

`dataStore.getUint(Keys.positionLastSrcChainId(positionKey))` is read twice — once for order assignment and again for flag assignment.

Recommendation

Cache the value locally to avoid redundant reads, similar to how it's done in ADL logic.

Resolution

GMX Team: Resolved.

L-16 | Frontrunning Risk

Category	Severity	Location	Status
Warning	Low	MarketPositionImpactPoolUtils.sol: 36-37	Acknowledged

Description

Since this function is executed via a timelock, there's potential for frontrunning or sandwiching. Although we didn't identify any major exploit paths during analysis, it remains a sensitive admin action with theoretical exposure to manipulation.

Refer to previous price deviation issues for examples

Recommendation

GMX should always execute this function using a frontrunning-resistant private RPC. This will reduce exposure and ensure consistent execution behavior.

Resolution

GMX Team: Acknowledged.

L-17 | Missing Safeguards

Category	Severity	Location	Status
Validation	Low	MultichainOrderRouterUtils.sol: 37-38	Resolved

Description

The `transferFeeFromOrderOrPosition` function allows users to reduce order collateral or update position collateral via fee deduction.

This makes it functionally similar to actions like `cancelOrder` or `decreaseOrder`. However, unlike those flows, it currently lacks important safeguards, including:

• `nonReentrant` (global)

• `FeatureUtils.validateFeature(...)`

These protections are standard across GMX's other state-changing functions and help ensure safety and feature control.

Recommendation

Add the same protections found in equivalent order-related flows to ensure consistent safety and behavior across user pathways.

Resolution

GMX Team: Resolved.

L-18 | Old Orders Can Overwrite Newer Ones

Category	Severity	Location	Status
Validation	Low	PositionUtils.sol: 784	Resolved

Description

The `updatePositionLastSrcChainId` is called during order execution and is used to save if the user interacted the last time from this chain or used a multichain account.

As the order could be a limit or stop order an old order could overwrite the last src chain id of a newer order.

For example:

• User created a limit order months ago

• User sees that the new multichain feature exists now and as he prefers to use another chain he sticks to using the system with a multichain account from now on and performs multiple market and limit orders from the multichain account

• The old limit order gets executed

• The system now thinks that the user prefers to receive tokens on the local chain

Recommendation

Only overwrite the `lastSrcChainId` if the order's creation timestamp is more recent than the last one

Resolution

GMX Team: Resolved.

L-19 | Integration Ids Can't Be Set With Multichain

Category	Severity	Location	Status
Logical Error	Low	SubaccountRouter.sol: 96	Acknowledged

Description

Multichain users are able to grant sub account approval by signing a struct. Additionally, the `removeSubaccount` is available to remove this approval using multichain.

However, the new `setIntegrationId` is only available in non-gasless transactions, using the `SubaccountRouter`.

Recommendation

Consider adding a `setIntegrationId` to the gasless sub account routers.

Resolution

GMX Team: Acknowledged.

L-20 | Validate Receiver In batchClaimAffiliateRewards

Category	Severity	Location	Status
Validation	Low	ReferralUtils.sol: 116	Resolved

Description

The `validateReceiver` function is not called during `batchClaimAffiliateRewards`, unlike in other claim flows such as `batchClaimFundingFees` or `batchClaimCollateral`.

Recommendation

Validate the receiver in `batchClaimAffiliateRewards` as well.

Resolution

GMX Team: Resolved.

L-21 | Impact Pool Withdrawals May Constantly Revert

Category	Severity	Location	Status
Informational	Low	TimelockConfig.sol: 343	Acknowledged

Description

The time lock admin can trigger a `signalWithdrawFromPositionImpactPool`, to withdraw a specific amount from the position impact pool.

This will create a delayed execution of the given withdrawal. Currently, this delay is set to 1 day (86400 seconds).

Therefore, any positive impact received by traders during this timeframe will reduce the position impact pool, causing reverts during the withdrawal execution if pool amount is insufficient.

Additionally, this admin function withdraws from the impact pool in both long and short tokens. Although markets will normally have sufficient token balances, there could be cases when there is only one token available, making the transaction fail.

Recommendation

Consider these edge cases when signaling the withdrawal to avoid reverts during execution.

Resolution

GMX Team: Acknowledged.

L-22 | Risk Free Trade By Causing Reverts

Category	Severity	Location	Status
Validation	Low	MultichainOrderRouterUtils.sol: 37-38	Resolved

Description

GMX introduced a feature called `transferFeeFromOrderOrPosition`, which allows users to use collateral from their existing order or position to pay fees in multichain orders. However, this feature introduces a critical vector for risk-free trade.

Attack Scenario:

1. An attacker opens a position, and also creates a dummy limit order that is not executable.

2. The attacker then creates a market increase order.

3. If the price moves unfavorably before execution, the attacker invokes an update or cancel action on the dummy limit order, using the position's collateral to pay the fee.

4. As a result, the market order fails to execute due to insufficient collateral.

5. The attacker successfully avoids downside risk without any cost — enabling risk-free trade behavior.

This attack works because collateral for the market order is silently drained through an unrelated limit order, and GMX does not block this cross-order collateral usage.

Recommendation

Add a check inside `transferFeeFromOrderOrPosition` to revert the if there are any preexisting open market order associated with the same position.

Resolution

GMX Team: Resolved.

L-23 | Incorrect Gas Tracking For Relay

Category	Severity	Location	Status
Logical Error	Low	BaseGelatoRelayRouter.sol, MultichainOrderRouter.sol, & MultichainClaimsRouter.sol	Resolved

Description

`withRelay()`, `withRelayForOrders()`, & `withRelayForClaims()` all declare `WithRelayCache` in memory prior to the call to track gas. Although it is initiated as empty, solidity will still allocate space for the array in memory.

This will lead to gas usage that is untracked, which will be paid for by GMX instead of the user. This leads to any call to a multi-chain or gasless feature to continuously cost GMX.

Recommendation

Move the declaration for `WithRelayCache` to after starting gas is assigned.

Resolution

GMX Team: Resolved.

L-24 | Frontrunning withdrawFromPositionImpactPool

Category	Severity	Location	Status
Warning	Low	MarketPositionImpactPoolUtils.sol	Acknowledged

Description

A new withdrawFromPositionImpactPool function has been added to allow token withdrawals from the impact pool. However, this function is time-locked and not atomic. Additionally, the withdrawal amount is determined at the time the time lock is scheduled.

Users are aware of when this action will be executed. Since positive price impact is capped based on the impact pool amount, users with pending positive impact will decrease their positions before the timelock execution to ensure they receive their rebates without being affected by the cap.

This not only creates race conditions and unfair advantages between users, but will also cause the timelock execution to fail due to a reduction in the available impact pool amount.

Additionally, since the amount is determined at the time of scheduling, but the available amount check is performed at the time of execution, the function might fail due to changes in the price impact pool after regular user trades in this time period.

Recommendation

Beware of this possibility, and if considered likely:

Consider validating the withdrawal amount and deducting it from the price impact pool at the time of scheduling, effectively reserving this amount for withdrawal until the time lock execution. Alternatively, consider performing this action atomically without a timelock.

Resolution

GMX Team: Acknowledged.

L-25 | Unnecessary srcChainId In bridgeIn

Category	Severity	Location	Status
Events	Low	MultichainTransferRouter.sol: 23	Resolved

Description

The `bridgeIn` function accepts a user-provided `srcChainId` as a parameter. However, this function is only used for same-chain transfers from the user's EOA to the multichain balance.

Therefore, the `srcChainId` is unnecessary. Additionally, this arbitrary `srcChainId` will be used in the `multiChainTransferIn` event, causing misleading event emissions for external listeners.

Recommendation

Consider removing the `srcChainId` for same-chain `bridgeIn` transfers.

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