GA GUARDIAN

GMX

GMX Crosschain V2.2 3

Security Assessment

July 26th, 2025

Summary

Audit Firm Guardian

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

Client Firm GMX

Final Report Date July 26, 2025

Audit Summary

GMX engaged Guardian to review the security of GMX Crosschain architecture. From the 19th of May to the 26th of May, a team of 5 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	3	0	0	0	0	3
High	6	0	0	0	0	6
Medium	13	0	0	4	0	9
Low	48	0	0	24	0	24
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	Anyone Can Drain The Multichain Balance	Validation	Critical	Resolved
C-02	Anyone Can Drain Multichain GM Token's Balance	Validation	Critical	Resolved
C-03	Arbitrary Execution & Balance Drain Risk	Logical Error	Critical	Resolved
H-01	Wrong Token Is Used In bridgeOutFromController	Logical Error	High	Resolved
H-02	Overflow DoS In withdrawFromPositionImpactPool	DoS	High	Resolved
H-03	Pending Amounts Errantly Capped To Zero	Logical Error	High	Resolved
H-04	Positive Impact Becomes Unbacked	Logical Error	High	Resolved
H-05	Max Impact Caps Double Count Lent Amounts	Logical Error	High	Resolved
H-06	Unclear Lending Constraints	Logical Error	High	Resolved
M-01	Unnecessary Relay Fee Charged	Logical Error	Medium	Resolved
M-02	Untrusted srcChainId	Logical Error	Medium	Resolved
M-03	Referrals Controlled For Smart Contract Traders	Access Control	Medium	Acknowledged
M-04	Composed Deposit Censoring	Frontrunning	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-05	Inaccurate Liquidations	Logical Error	Medium	Acknowledged
M-06	Incorrect Atomic Oracle Provider Handling	Logical Error	Medium	Resolved
M-07	Max Lent Bypassed By Splitting Orders	Warning	Medium	Resolved
M-08	Impact Pool Withdrawals Use An Invalid Ratio	Logical Error	Medium	Resolved
M-09	Pending Price Impact Can Be Lost	Validation	Medium	Resolved
M-10	Gas Limits Underestimated	Configuration	Medium	Acknowledged
M-11	Malicious Token Gas Griefs Keepers	Warning	Medium	Resolved
M-12	Impossible To Execute Some Timelock Actions	Logical Error	Medium	Resolved
M-13	Subaccounts May Divert Multichain Funds	Unexpected Behavior	Medium	Acknowledged
L-01	External Call Gas Is Not Corrected	Logical Error	Low	Acknowledged
L-02	MarketPoolValueInfo NatSpec Wrong	Informational	Low	Resolved
L-03	bridgeIn Is Marked As Payable	Best Practices	Low	Acknowledged
L-04	Duplicated Code	Best Practices	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-05	MessagingReceipt Not Returned	Best Practices	Low	Acknowledged
L-06	Unnecessary referralStorage Variable	Superfluous Code	Low	Acknowledged
L-07	Unnecessary Ternary	Gas Optimization	Low	Resolved
L-08	Lendable Amt May Not Belong To Pool	Logical Error	Low	Acknowledged
L-09	Price Impact Not Guaranteed	Logical Error	Low	Acknowledged
L-10	Incorrect Zero Amount Check In recordBridgeIn	Validation	Low	Resolved
L-11	Early Return Should Include Equality Operator	Gas Optimization	Low	Resolved
L-12	Inconsistent Handler Addresses	Configuration	Low	Acknowledged
L-13	Incorrect Comparison Operator	Logical Error	Low	Resolved
L-14	Missing MAX_DATA_LENGTH Initialization	Configuration	Low	Resolved
L-15	eidToSrcChainId Might Not Be Defined	Configuration	Low	Resolved
L-16	Modifier Gas Not Accounted For	Logical Error	Low	Acknowledged
L-17	Features Not Validated During Shifts	Validation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-18	Inability To Bridge Out During Shift	Logical Error	Low	Resolved
L-19	Missing nonReentrant Protection	Validation	Low	Resolved
L-20	Misspelled Filenames In Executor Contracts	Best Practices	Low	Resolved
L-21	Potential Feed ID Collision Due To Truncation	Validation	Low	Acknowledged
L-22	Old Orders Can Overwrite Newer Ones	Validation	Low	Resolved
L-23	Missing Validation For bridgeOutFromController	Validation	Low	Resolved
L-24	LayerZero Configuration	Configuration	Low	Acknowledged
L-25	isSrcChainIdEnabledKey Can Be Bypassed	Validation	Low	Resolved
L-26	Keepers May Censor Actions Using BridgeOut	Censoring	Low	Acknowledged
L-27	Lacking Stargate Fee Protection	Warning	Low	Acknowledged
L-28	Missing NatSpec Param	Documentation	Low	Resolved
L-29	Dangerous Timelock Admin Assignment	Best Practices	Low	Acknowledged
L-30	Misleading Timelock Transaction Success	Unexpected Behavior	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-31	Outdated TimelockController Used	Best Practices	Low	Acknowledged
L-32	Unfavorable Impact Withdrawal Rounding	Rounding	Low	Resolved
L-33	Lacking Default Admin Rules	Best Practices	Low	Acknowledged
L-34	Unnecessary amountLD Param In recordBridgeIn	Informational	Low	Resolved
L-35	Inaccurate executionPrice Emitted	Events	Low	Acknowledged
L-36	Impact Distributions For Pending Impact	Warning	Low	Acknowledged
L-37	maxLendableFactor Configuration	Configuration	Low	Resolved
L-38	Price Impact Griefing	Warning	Low	Acknowledged
L-39	Stargate Quote Inaccuracies	Warning	Low	Acknowledged
L-40	Lacking Collateral Factor Validations	Validation	Low	Acknowledged
L-41	Asymmetric Capping	Warning	Low	Acknowledged
L-42	Functions Cannot Be Used With Multicall	Warning	Low	Acknowledged
L-43	Missing isSrcChainIdEnabled Check	Access Control	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-44	Price Impact Factors Should Be Adjusted	Warning	Low	Acknowledged
L-45	Redundant WithRelayCache Struct	Informational	Low	Resolved
L-46	Initialize Frontrunning	Frontrunning	Low	Resolved
L-47	Unclear Native Deposit BridgeOut Validation	Validation	Low	Resolved
L-48	Unexpected Deposit Execution Reverts	Validation	Low	Resolved

C-01 | Anyone Can Drain The Multichain Balance

Category	Severity	Location	Status
Validation	● Critical	LayerZeroProvider.sol: 85-86	Resolved

Description

GMX has added a new optional functionality in `lzCompose`, allowing users to instantly deposit their bridged tokens in exchange for GM or GLV tokens.

However, since GMX relies on the decoded account from the compose message, anyone can spoof any other user.

Previously, this wasn't an issue—if someone used account B instead of their own account A, they would simply end up donating their bridged tokens to account B's multichain balance.

But with the introduction of `handleDepositFromBridge/_handleGlvDepositFromBridge`, which accepts `actionData` as another user-controlled field, account B can now spoof account A and use A's multichain balance to perform a GM or GLV deposit, with themselves (B) as the receiver.

Recommendation

Consider validating that the action to be executed for a given account is authorized via a signature from that account, using a digest of the `actionData`.

Resolution

GMX Team: Resolved.

C-02 | Anyone Can Drain Multichain GM Token's Balance

Category	Severity	Location	Status
Validation	● Critical	MultichainGlvRouter.sol: 48-49	Resolved

Description

GMX added `createGlvDepositFromBridge` in `MultichainGlvRouter` to facilitate `_handleGlvDepositFromBridge`. However, this function is missing the `onlyController` modifier.

As a result, anyone can trigger a GLV deposit on behalf of any user, with themselves as the receiver—since this call is not validated using a signature.

Recommendation

Consider adding the `onlyController` modifier to `createGlvDepositFromBridge` to restrict access to authorized callers only.

Resolution

GMX Team: Resolved.

C-03 | Arbitrary Execution & Balance Drain Risk

Category	Severity	Location	Status
Logical Error	● Critical	ExecuteDepositUtils.sol: 277-278	Resolved

Description

`bridgeOutFromController` has been added to `executeDeposit` to allow users to instantly bridge their received GM tokens to a chosen chain. However, this function also carries the `withRelay` modifier when used via the transfer router, enabling arbitrary external calls or sends.

Since `bridgeOutFromController` uses `deposit.receiver()` as the input or payee for `withRelay`, this opens up a way for an attacker to drain another user's multichain balance. For example, attacker A can call the function using depositor B as the receiver, effectively spending B's funds with arbitrary logic via `withRelay`.

Additionally, using `withRelay` here introduces other problems:

1. Since `msg.sender` won't be Gelato, the call is treated as sponsored, and users will be unnecessarily charged a Gelato fee — something they already cover through GMX's keeper fees.

2. This `withRelay` usage also enables risk-free deposits with conditional execution:

`withRelay` hook allows arbitrary external calls, and since gas cost can vary depending on the logic, this allows user to manipulate gas consumption to cause conditional reverts.

3. Since oracle prices are preset in `executeDeposit`, any modification or re-setting of those prices during `handleRelayBeforeAction` (if attempted) could cause the transaction to revert.

Recommendation

Consider removing the `withRelay` modifier from `bridgeOutFromController` entirely, as it doesn't appear to serve a valid use case in this context and introduces multiple risks.

Resolution

GMX Team: Resolved.

H-01 | Wrong Token Is Used In bridgeOutFromController

Category	Severity	Location	Status
Logical Error	High	ExecuteGlvDepositUtils.sol: 142-144	Resolved

Description

The `bridgeOutFromController` function is introduced to bridge the GM and GLV tokens to the `srcChain` based on the provided `dataList`.

However, incorrect token and amount values are used when calling `bridgeOutFromController` in the `executeGlvDeposit` function during the GLV deposit flow.

The `market` token is used instead of the `glv` token, and the `receivedMarketTokens` value is used instead of `mintAmount`.

Recommendation

Use the `glv` token and `mintAmount` when calling this function during the GLV flow.

Resolution

GMX Team: Resolved.

H-02 | Overflow DoS In withdrawFromPositionImpactPool

Category	Severity	Location	Status
DoS	High	MarketPositionImpactPoolUtils.sol: 77-82	Resolved

Description

The pending impact amount is now globally tracked with the `totalPendingImpactAmountKey` as a `int256` value. During position increase, the `priceImpactAmount` is applied as delta to this global value.

However, `priceImpactAmount` can be either positive or negative, depending on the open interest values of the market.

During `withdrawFromPositionImpactPool`, the `getTotalPendingImpactAmount` is read from storage and converted to `uint256` so it can be compared to `adjustedImpactPoolAmount`.

Consequently, this call will revert every time the total impact amount is negative, DoS'ing impact pool withdrawals.

Recommendation

Consider validating if `totalPendingImpactAmount` is positive before converting it to `uint256`. The sufficiency check is not required for negative pending impact amounts, as those are meant to be paid by users at position decrease.

Resolution

GMX Team: Resolved.

H-03 | Pending Amounts Errantly Capped To Zero

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol	Resolved

Description

In the capPositiveImpactUsdByPositionImpactPool function if the totalImpactPoolAmount value is negative the resulting capped impact is immediately set to 0. However this prevents traders from claiming their positive pending amounts when there is indeed some portion of backing value in the impact pool to cover it.

Consider the following scenario:
• Trader A has a pending impact of -5
• Trader B has a pending impact of 3
• The total pending impact is -2
• Trader A closes their position and receives positive impact of 6
• The impact pool amount is 4 at time 0
• The computed totalImpactPoolAmount value for Trader A at time 0 is 4 (impact pool amount) - 3 (totalPending - Trader A pending) = 1
• At time 2 the impact pool distributes down to 2.99
• The computed totalImpactPoolAmount value for Trader A at time 2 is 2.99 (impact pool amount) - 3 (totalPendingImpactAmount, -2 + 5) = -0.01
• The impact is capped to 0, when in reality the amalgam of impact assets could cover a portion of the positive impact: 2.99 (impact pool amount) - 2 (pending impact amount) = 0.99 coverable of the 1 net impact.

Recommendation

The crux of this issue is an over complicated handling of the pending impact amount during the capping process. Pending amounts were capped already on increase and can be considered as effectively a part of the impact pool. The current capPositiveImpactUsdByPositionImpactPool function includes the pending impact as a part of the impact that is capped on decrease, and then corrects for this by removing the positionProportionalPendingImpactAmount from the totalPendingImpactAmount.

However this action makes an underlying flawed assumption, that the positionProportionalPendingImpactAmount will be simply removed from the amalgamation of the impact pool, pending impact, and lentAmount. Instead the positionProportionalPendingImpactAmount just transfers which variable it is tracked in but stays within the amalgamation of these three variables which make up the totality of the impact pool. To remove this edge case and others which possibly exist but are hard to analyze, re-implement the capping of impact on decrease in the following way:

• The pending impact amount should be treated as already realized and not capped by the impact amalgamation variables
• The pending impact now does not need to be corrected for in the totalPendingImpactAmount as it already belongs to the impact amalgamation and will stay there

Resolution

GMX Team: Resolved.

H-04 | Positive Impact Becomes Unbacked

Category	Severity	Location	Status
Logical Error	High	DecreasePositionCollateralUtils.sol	Resolved

Description

The lentAmount logic is meant to enable positive impact to be realized when there is no immediate amount available in the position impact pool.

The lentAmount becomes a loan against a corresponding amount of negative impact that is pending for another position. However this loan is made against "collateral" that can disappear on decrease.

In the event that a position with a large negative pending impact is decreased and has it's negative impact capped by the maxPriceImpactFactor, the negative pending impact is essentially erased.

As a result it is possible for the lent amounts to become more than the value of the total pending impact.

Recommendation

When a position is decreasing and realizing a negative pending impact, require that the position pays a portion of the lentAmount that it owes rather than capping it freely. Otherwise consider removing positive price impact for markets which will have their negative impact capped.

Resolution

GMX Team: Resolved.

H-05 | Max Impact Caps Double Count Lent Amounts

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol: 947	Resolved

Description

In the capPositiveImpactUsdByPositionImpactPool function the totalImpactPoolAmount is computed as the total net impact token amount available between the impact pool and the pending impact amounts. This amount represents the total impact amount available assuming that all traders with pending impact settle their positions.

When a trader receives positive impact and realizes it, withdrawing from the impact pool, the impact pool amount is deducted and that is therefore reflected in the lowering of this cap. However, when a trader receives positive impact and realizes it, withdrawing instead against the pending impact amount and creating a lentAmount, neither the impact pool amount or the pending impact amounts are reduced to reflect a lowering of the cap.

Consider the following example:
• Start with a new market
• Trader A opens long size 100 and receives -10 pending impact
• Trader B opens short size 100 and receives 10 pending impact
• The impact pool is still 0
• Trader C opens short size 100 and receives -10 pending impact
• Trader B closes their position and realizes 10 new impact and their 10 pending impact
• The total impact cap is 20 from A and C, and B's pending impact is removed from the totalPendingImpactAmount
• B receives all 20 impact and the lent amount is increased to 20
• On any future caps the maxPriceImpactUsd reports that the full 20 impact amount from A and C's pending impact can be used, however it is already lent to B.

Recommendation

Deduct the cache.lentUsd in the cache.maxPriceImpactUsd calculation like so:

```
cache.maxPriceImpactUsd = cache.totalImpactPoolAmount * prices.indexTokenPrice.min.toInt256() - cache.lentUsd;
```

Resolution

GMX Team: Resolved.

H-06 | Unclear Lending Constraints

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol: 897-898	Resolved

Description

The `capPositiveImpactUsdByPositionImpactPool` function is designed to cap the positive price impact a user can realize. It takes into account a market-configured `maxLendable` amount, which represents how much price impact the protocol can “lend” to users even when the impact pool is empty.

This design introduces the concept of "lent price impact" — a soft mechanism that allows the system to temporarily cover a user’s positive price impact beyond the available impact pool, under the assumption that future negative price impact (pending) will compensate for it.

However, the function does not enforce any constraint that this lent impact must be matched by existing or eventual pending price impact. It allows users to borrow up to `maxLendable`, regardless of whether the system has enough pending impact to support it, or whether all positions could close without offsetting it.

Recommendation

Due to the current behavior, it’s unclear what the intended design philosophy behind “lent” price impact is. There appear to be two possible interpretations:

1. Lent as a free-floating loan against `C` (pool value) up to some ratio
In this model, pending impact is expected to eventually pay it back, but there is no enforced relation or constraint. Users are allowed to borrow up to `maxLendable` regardless of whether sufficient pending impact exists.
→ This is flexible but introduces systemic risk if pending impact never materializes.

2. Lent must be backed by pending price impact
In this stricter model, you can only borrow positive price impact if there is matching (realized or pending) negative price impact. Lending and pending are directly tied.
→ This ensures solvency but is more conservative.

If Option 2 is the intended model, consider enforcing caps on positive impacts using pending amounts directly, rather than relying solely on the `maxLendable` ratio. This would ensure that all lent amounts are matched by actual system liabilities, maintaining consistency and preventing impact pool deficits.

Resolution

GMX Team: Resolved.

M-01 | Unnecessary Relay Fee Charged

Category	Severity	Location	Status
Logical Error	Medium	MultichainGmRouter.sol: 56-57	Resolved

Description

GMX now allows users to instantly deposit their bridged tokens using `createDepositFromBridge` and `createGlvDepositFromBridge`. However, both of these functions are wrapped with the `withRelay` modifier.

During execution, this causes `handleRelayFee` to treat the call as sponsored, since `msg.sender` is not Gelato. As a result, users are forced to pay a Gelato relay fee unnecessarily.

Recommendation

Consider implementing a modified `withRelay` variant specifically for bridge-based deposits. This version should:

• Only account for deposit-specific keeper fees and collateral.

• Avoid charging an additional Gelato relay fee

This would prevent redundant fees and align the cost model with user expectations for bridged deposit

Resolution

GMX Team: Resolved.

M-02 | Untrusted srcChainId

Category	Severity	Location	Status
Logical Error	Medium	LayerZeroProvider.sol	Resolved

Description

In the `lzCompose` function the `srcChainId` is decoded directly from the `composeMessage` which is entirely controlled by the sender.

As a result, this value may not accurately reflect the chain that the message was bridged from and can be used to bypass the `isSrcChainIdEnabled` validations.

Recommendation

Instead, the trusted `srcEid` included in the message in the `lzCompose` call should be used to infer the source chain.

This can be retrieved using the `OFTComposeMsgCodec.srcEid` function. If desired the `srcEid` can be mapped back to a `srcChainId`.

Resolution

GMX Team: Resolved.

M-03 | Referrals Controlled For Smart Contract Traders

Category	Severity	Location	Status
Access Control	Medium	MultichainSender.sol	Acknowledged

Description

The `MultichainSender` contract allows the controller of an address on one chain to set the referral code for the owner of that same address on Arbitrum.

In the case of Smart Contract accounts, the controller of this address may be different across chains. As a result an untrusted party is able to control the referral code used for the target account.

Recommendation

Consider requiring that referral codes be set by a signature for multichain actions, therefore ensuring that the account is an EOA. For accounts that are smart contracts on Arbitrum, they can set their referral code on Arbitrum itself.

Resolution

GMX Team: Acknowledged.

M-04 | Composed Deposit Censoring

Category	Severity	Location	Status
Frontrunning	Medium	LayerZeroProvider.sol	Resolved

Description

The lzCompose function can be called by any actor after the configured DVNs have verified the message.

As a result an actor can invoke the lzCompose function on the LayerZero endpoint and provide an insufficient amount of gas to fully execute the lzCompose received funds accounting and the _handleDepositFromBridge or _handleGlvDepositFromBridge action that follows.

This would allow the malicious executor to cause the createDepositFromBridge or createGlvDepositFromBridge calls to run out of gas during execution and enter the catch block which they are wrapped in.

The affected user would then have to create another deposit action through the MultichainGmRouter contract.

Recommendation

Consider validating that the amount of gas that has been provided is sufficient before calling the createDepositFromBridge or createGlvDepositFromBridge functions.

Resolution

GMX Team: Resolved.

M-05 | Inaccurate Liquidations

Category	Severity	Location	Status
Logical Error	Medium	PositionUtils.sol: 320-321	Acknowledged

Description

GMX currently fails to correctly apply caps for both positive and negative price impacts in `isPositionLiquidatable`, which can incorrectly return `false` when a position should be liquidated.

Positive Price Impact:
• GMX does not cap the positive price impact to the available amount in the `impactPoolAmount`.
• As a result, a position might appear solvent based on a theoretical positive price impact, even though GMX cannot actually pay the profit due to insufficient funds in the impact pool.
• This leads to positions avoiding liquidation despite being undercollateralized in practice.

Negative Price Impact:
• Due to the new deferred accounting of price impacts (i.e., unrealized/pending impact on position increase), users can open positions with a large negative price impact that is not immediately realized.
• During liquidation checks, `isPositionLiquidatable` computes the total price impact and applies the `maxNegativePriceImpactUsd` cap to it.
• However, this cap was originally meant to protect against sudden adverse price movements, not unrealized negative impact from user's own position open.
• By applying the cap to the total impact (including the user's pending impact), the function can understate the risk and allow a user to avoid liquidation.

Recommendation

• Positive Price Impact: Cap it to `impactPoolAmount` during liquidation checks. `capPositiveImpactUsdByPositionImpactPool`
• Negative Price Impact: Only cap new negative price impact incurred from current price slippage during liquidation, not the previously accrued/pending impact from position opening.

Resolution

GMX Team: Acknowledged.

M-06 | Incorrect Atomic Oracle Provider Handling

Category	Severity	Location	Status
Logical Error	Medium	Oracle.sol	Resolved

Description

The atomic oracle provider validation is intended to be separated from the reference oracle check and timestamp adjustment logic.

However instead of separating this logic for the timestamp adjustment, the timestamp adjustment is still always applied based on the `isAtomicProvider` status.

Furthermore, if an action is `forAtomicAction` and the provider `shouldAdjustTimestamp` value is assigned as `true` then the oracle reverts with `NonAtomicOracleProvider`.

Recommendation

Replace the initial atomic oracle validation with:

```
if (forAtomicAction) {
if (isAtomicProvider) {
revert Errors.NonAtomicOracleProvider(_provider);
}
}
```

And replace the timestamp adjustment logic with:

```
if (provider.shouldAdjustTimestamp()) {
uint256 timestampAdjustment = dataStore.getUint(Keys.oracleTimestampAdjustmentKey(_provider,
token));
validatedPrice.timestamp = timestampAdjustment;
}
```

Resolution

GMX Team: Resolved.

M-07 | Max Lent Bypassed By Splitting Orders

Category	Severity	Location	Status
Warning	Medium	MarketUtils.sol: 948	Resolved

Description

The maxLent validation serves as a sanity check to cap the price impact by a proportion of the underlying market short and long tokens.

However this check can be easily circumvented for increase orders by splitting up a large order into several smaller orders so the price impact on any given order is smaller than the maxLendableUsd based on the backing token amounts in the market.

Consider the following scenario:
• A GM market has 100 backing long tokens and 100 backing short tokens
• User A has an order that will experience 30 positive impact
• The maxLendableUsd value is 20
• User A splits their increase order up into two halves which each experience 15 positive impact
• The execution of the first increase order has no effect on the maxLendable for the second order since the impact pool is not adjusted for the impact amount on increase
• Therefore User A is able to bank their 30 positive impact as a pending impact amount when otherwise they would have only been able to save 20

This gives informed users who can abuse this behavior an advantage over those who are unaware in order to accumulate more aggregate price impact.

Recommendation

Consider if the max lendable validation should be applied on increase. If it shouldn't be then consider only applying the maxLendable validation on decrease to prevent this gaming.

Resolution

GMX Team: Resolved.

M-08 | Impact Pool Withdrawals Use An Invalid Ratio

Category	Severity	Location	Status
Logical Error	Medium	MarketPositionImpactPoolUtils.sol	Resolved

Description

The withdrawFromPositionImpactPool function removes long and short tokens at a 50/50 dollar value split. However in the majority of GM markets the backing token amounts will not be evenly distributed 50/50.

In many cases this removal can cause a further imbalance in a GM market or even revert due to unavailability of tokens.

Consider the following scenario:
• Our GM market has 10 token A and 20 token B
• Both token A and token B are valued at $1
• $10 is withdrawn with the withdrawFromPositionImpactPool function, a large value is used for explication
• 5 token A and 5 token B is removed from the GM market
• the GM market is left with 5 token A and 15 token B
• The imbalance in the market has gone from 33%/67% to 25%/75%

Because of this behavior, each withdrawFromPositionImpactPool action increases the amount of positive price impact that can be earned without providing negative impact into the impact pool, invalidating an important invariant of the GMX exchange.

Recommendation

The backing token amounts must be removed at the ratio of the backing tokens in the existing GM market, similarly to the GM withdrawal logic.

Resolution

GMX Team: Resolved.

M-09 | Pending Price Impact Can Be Lost

Category	Severity	Location	Status
Validation	Medium	MarketUtils.sol: 946-956	Resolved

Description

During position decrease, the `capPositiveImpactUsdByPositionImpactPool` is invoked with the `totalImpactUsd` and the `proportionalPendingImpactAmount`.

Previously, this function will only cap positive impact to the usd value of the impact pool tokens. However, with the introduction of `lentAmount`, the `maxPriceImpactUsd` will be capped at the `maxLendableUsd`.

Consequently, if the `maxLendableUsd` is lower than the `maxPriceImpactUsd` and there are no lent funds, user will receive a lower `priceImpactUsd`, even if the impact pool has enough funds to cover it.

Recommendation

Consider capping the `maxPriceImpactUsd` only when `lentUsd > 0`

Resolution

GMX Team: Resolved.

M-10 | Gas Limits Underestimated

Category	Severity	Location	Status
Configuration	Medium	Global	Acknowledged

Description

The current configured gas limit for GM and GLV deposits are 1,800,000 and 2,000,000 respectively. Additionally, the `minAdditionalGasForExecution` is set to 1,000,000.

However, this gas limit does not account for the optional feature of bridging out tokens after deposit execution. During execution, keeper's gas will be validated against these values using `GasUtils.validateExecutionGas`.

This check might be valid for deposits without the bridge out option, but the gas will not be enough if it contains the extra feature.

Although keepers may send more gas to account for this issue, the gas validation is incorrect and can make them experience Out of Gas errors.

Recommendation

Consider adjusting the `depositGasLimit` and `glvDepositGasLimit` to account for the gas spent in the new bridge out feature after deposit execution

Resolution

GMX Team: Acknowledged.

M-11 | Malicious Token Gas Griefs Keepers

Category	Severity	Location	Status
Warning	Medium	SwapUtils.sol	Resolved

Description

During an increase order execution, the provided `initialCollateral` token is not validated to be a valid collateral token associated with any market at the point of the `SwapUtils.swap` function when the initial `params.bank.transferOut` invocation is made.

As a result, the `tokenIn` could be a malicious token which returns malicious `returnData` which holds revert reason bytes which proclaim to be an enormously long string and thus force the keepers to expend a large amount of gas above and beyond what the original execution fee covers.

Recommendation

Consider validating if the initial token is a valid collateral token of the first market token in the `swapPath` in the `swap` function.

Resolution

GMX Team: Resolved.

M-12 | Impossible To Execute Some Timelock Actions

Category	Severity	Location	Status
Logical Error	Medium	ConfigTimelockController.sol	Resolved

Description

In the `ConfigTimelockController` contract the `executeWithOraclePrices` function hard-codes 0 as a predecessor and 0 as a salt.

As a result, any actions that have been scheduled with a salt or predecessor that require oracle prices cannot be executed through the `ConfigTimelockController`.

This proves especially problematic when the executor wishes to withdraw from the position impact pool of the same market.

The caller will be unable to withdraw the same amount to the same receiver since the id of this action will have already been marked as Done.

This immediately affects the `signalWithdrawTokens` function but may also apply to future use-cases of the timelock.

Recommendation

Add predecessor and salt parameters to the `executeWithOraclePrices` function so that transactions can be differentiated and ordered.

Resolution

GMX Team: Resolved.

M-13 | Subaccounts May Divert Multichain Funds

Category	Severity	Location	Status
Unexpected Behavior	Medium	Global	Acknowledged

Description

The `lastSrcChainId` is assigned for the position upon execution of every increase or decrease order. If the integration is not careful to handle funds being unexpectedly sent to the multichain balance rather than the expected native arbitrum `srcChainId`, then funds could potentially be trapped.

A sub account can maliciously redirect funds to the multichain balance by creating a market order with a multichain `srcChainId` shortly before the position is either liquidated or ADL'd. For smart contract accounts on Arbitrum this can lead to trapped funds.

Recommendation

Consider allowing integrations to explicitly set the chain id associated with their position with a `setSavedSrcChainId` function, similar to the `setSavedCallbackContract` function.

Resolution

GMX Team: Acknowledged.

L-01 | External Call Gas Is Not Corrected

Category	Severity	Location	Status
Logical Error	Low	DepositHandler.sol: 128	Acknowledged

Description

The executeDepositFromController function was introduced in the DepositHandler contract to reduce dependencies necessary across contracts. However this adds an additional external call in the deposit execution process during shifts.

This additional external call does not adjust the params.startingGas for the 63/64 rule and therefore would overestimate the amount of gas used for the deposit execution.

However the executionFee assigned for the deposit is assigned as zero, meaning that the gas calculations for the deposit are not made nor reflected in any fee charged.

Recommendation

No change is necessary since there is no net impact of the omission of the external call gas correction.

This finding serves merely to document this behavior in the event that future uses of the executeDepositFromController would include deposits that use a nonzero executionFee.

Resolution

GMX Team: Acknowledged.

L-02 | MarketPoolValueInfo NatSpec Wrong

Category	Severity	Location	Status
Informational	Low	MarketPoolValueInfo.sol	Resolved

Description

The `NatSpec` of the `MarketPoolValueInfo` function does not match the real variables.

Recommendation

Update the `NatSpec`.

Resolution

GMX Team: Resolved.

L-03 | bridgeIn Is Marked As Payable

Category	Severity	Location	Status
Best Practices	Low	MultichainTransferRouter.sol: 30	Acknowledged

Description

The `bridgeIn` function is marked as payable, but there are no functions that use native currency or handle refunds. Therefore, any native tokens sent with this functions will be permanently lost.

Recommendation

Remove the payable modifier to avoid loss of funds. If this is used to save transaction gas, consider documenting this to users.

Resolution

GMX Team: Acknowledged.

L-04 | Duplicated Code

Category	Severity	Location	Status
Best Practices	Low	SubaccountRouter.sol	Resolved

Description

In the `SubaccountRouter.createOrder` function the validations on the receiver and cancellation receiver are repetitive with the implementation of `validateCreateOrderParams`.

The only difference being that in the `SubaccountRouter.createOrder` function the `InvalidReceiverForSubaccountOrder` error is used, while in the `validateCreateOrderParams` function the `InvalidReceiver` error is used.

Recommendation

Consider deduplicating this validation so that in the future any adjustments to the `validateCreateOrderParams` validation will not be missed from the validation in the `SubaccountRouter.createOrder` function.

Furthermore, consider using the `InvalidReceiverForSubaccountOrder` error in the `validateCreateOrderParams` function.

Resolution

GMX Team: Resolved.

L-05 | MessagingReceipt Not Returned

Category	Severity	Location	Status
Best Practices	Low	MultichainSender.sol: 39	Acknowledged

Description

The `MultichainSender` will allow users to send a message through Layer Zero. However, the `sendMessage` function does not return the `MessagingReceipt` param, which is available in the value returned from `_lzSend`. This will a nice feature to have for integrations and UI/UX

Recommendation

Return the `MessagingReceipt` in the `sendMessage` function.

Resolution

GMX Team: Acknowledged.

L-06 | Unnecessary referralStorage Variable

Category	Severity	Location	Status
Superfluous Code	Low	MultichainOrderRouter.sol	Acknowledged

Description

There is no implemented use for the `referralStorage` variable in the `MultichainOrderRouter` contract.

Recommendation

Consider either implementing the use-case for the `referralStorage` variable or removing it from the `MultichainOrderRouter` contract.

Resolution

GMX Team: Acknowledged.

L-07 | Unnecessary Ternary

Category	Severity	Location	Status
Gas Optimization	Low	LiquidationUtils.sol	Resolved

Description

In the `LiquidationUtils` library the boolean expression `cache.lastSrcChainId = 0 true : false` is used, however the `cache.lastSrcChainId = 0` value can be used directly.

Recommendation

Remove the unnecessary ternary operator.

Resolution

GMX Team: Resolved.

L-08 | Lendable Amt May Not Belong To Pool

Category	Severity	Location	Status
Logical Error	Low	MarketUtils.sol: 933-936	Acknowledged

Description

In the `capPositiveImpactUsdByPositionImpactPool` function the `maxLendableUsd` variable is calculated by applying a factor on the value of the tokens in the pool. This does not factor in PnL, borrowing fees, etc.

This could lead to giving out funds that the pool doesn't actually own which could for example lead to users not being able to close their positions as there are currently not enough funds in the pool to do so. This is especially dangerous in new pools which do not have accrued much liquidity yet.

Recommendation

Consider working with the `getPoolValueInfo` function instead of the value of the pool amounts.

Resolution

GMX Team: Acknowledged.

L-09 | Price Impact Not Guaranteed

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

Price impact from increase is still not 100% guaranteed. The reason for this are the following:

• The price impact amount is reserved now but the price of these tokens can change between increase and decrease

• The lending capacity limited by the `maxLendableImpactFactor` can be exhausted

• the price impact pool is distributed to the pool over time

Recommendation

Rethink or acknowledge and document this behaviour.

Resolution

GMX Team: Acknowledged.

L-10 | Incorrect Zero Amount Check In recordBridgeIn

Category	Severity	Location	Status
Validation	Low	MultichainUtils.sol: 40	Resolved

Description

The `lzCompose` will record bridged funds using `MultichainUtils.recordBridgeIn`. Although the function reverts if `amount = 0`, this check is done on the param passed, not on the actual recorded amount.

Therefore, if `multichainVault.recordTransferIn(token)` records zero amount, the transaction won't revert. The `lzCompose` will succeed but user will not receive the tokens.

Recommendation

Consider validating zero amounts with the value returned from the `recordTransferIn` function.

Resolution

GMX Team: Resolved.

L-11 | Early Return Should Include Equality Operator

Category	Severity	Location	Status
Gas Optimization	Low	MarketUtils.sol: 926	Resolved

Description

The `capPositiveImpactUsdByPositionImpactPool` function will early return if the `totalImpactPoolAmount` is less than 0. This means that there are no available funds in the impact pool to pay for more positive impact.

However, if `totalImpactPoolAmount = 0`, it will waste some gas calculating the other parameters, as the `maxPriceImpactUsd` will also be zero, and the final check will cap the `priceImpactUsd` to 0.

Recommendation

Add the equality operator to the early return check:

```
if (cache.totalImpactPoolAmount = 0) {
return 0;
}
```

Resolution

GMX Team: Resolved.

L-12 | Inconsistent Handler Addresses

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

Certain library calls were refactored into using external calls to handlers. New state variables were added to these handlers to store contracts like `swapHandler`, `multichainTransferRouter`, `depositHandler`, `withdrawalHandler`, etc.

Therefore, deploying a new handler (i.e. `swapHandler`) will require the other handlers to be updated. However, there are no setter functions for this update, which will require a re-deploy of all contracts that new handler.

This may also cause some issues when contracts use different handler or router versions.

Recommendation

Document this behavior and make sure contracts are re-deployed for new handler/router updates, with the appropriate role revoking and granting.

Resolution

GMX Team: Acknowledged.

L-13 | Incorrect Comparison Operator

Category	Severity	Location	Status
Logical Error	Low	MarketPositionImpactPoolUtils.sol: 93	Resolved

Description

The `withdrawFromPositionImpactPool` function includes some sanity checks to make sure the withdrawn funds do not exceed the impact pool amount, accounting for the `totalPendingImpactAmount`.

However, the transaction will revert if the amount to withdraw is exactly the same as the available in the impact pool:

```
if (adjustedImpactPoolAmount <= amount) {
revert Errors.InsufficientImpactPoolValueForWithdrawal(amount,
poolValueInfo.impactPoolAmount, totalPendingImpactAmount);
}
```

Recommendation

Consider replacing the comparison operator `<=` for `<` to avoid unexpected reverts.

Resolution

GMX Team: Resolved.

L-14 | Missing MAX_DATA_LENGTH Initialization

Category	Severity	Location	Status
Configuration	Low	Global	Resolved

Description

The `MAX_DATA_LENGTH` is added as an allowed key but its value is never initialized. Therefore, any order created with a `dataList` length greater than zero will revert.

Recommendation

Consider adding `MAX_DATA_LENGTH` configuration to deploy script and avoid failed order creations.

Resolution

GMX Team: Resolved.

L-15 | eidToSrcChainId Might Not Be Defined

Category	Severity	Location	Status
Configuration	Low	LayerZeroProvider.sol: 172	Resolved

Description

The `eidToSrcChainId` will convert LZ ids to actual chain ids. However, this mapping of ids needs to be implemented by the config keeper. If user bridges out to a `dstEid` that is not defined in GMX data store, a `srcChainId` of 0 will be used.

Although this issue only impacts events, it may mislead off chain services and data analytics, if a value of zero is used for `srcChainId`.

Recommendation

Make sure all LZ eids are mapped to the corresponding src chain ids or at least the ones supported by Stargate. Alternatively, consider reverting if the `srcChainId` is zero to make sure only the GMX supported Eids are used.

Resolution

GMX Team: Resolved.

L-16 | Modifier Gas Not Accounted For

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

The new controller functions in the Multichain routers contain three modifiers in the following order:

`nonReentrant`, `onlyController`, `withRelay`

Therefore, the `startingGas` cached in `withRelay` will not account for the gas used in `onlyController`.

Recommendation

Consider moving `onlyController` to the last position in the list of modifiers.

Resolution

GMX Team: Acknowledged.

L-17 | Features Not Validated During Shifts

Category	Severity	Location	Status
Validation	Low	ShiftUtils.sol: 151	Resolved

Description

The shift flow requires a withdrawal from one market and a deposit to another, using `executeDepositFromController` and `executeWithdrawalFromController`.

These functions only validate if sender has a controller role, but not if the actual deposit and withdrawal feature are enabled.

Recommendation

Validate deposit and withdrawal feature in the respective controller function.

Resolution

GMX Team: Resolved.

L-18 | Inability To Bridge Out During Shift

Category	Severity	Location	Status
Logical Error	Low	ShiftUtils.sol: 287	Resolved

Description

The shift execution includes a withdrawal from a market and deposit to a new one. During the deposit flow of a shift, an optional `bridgeOutFromController` is executed, which will only early return if the `dataList` is not correctly formed.

However, during this shift deposit, the `multichainTransferRouter` is set as `address(0)`. This causes all shift executions to revert if `dataList` contains the `Keys.GMX_DATA_ACTION`.

Recommendation

If this is the expected behavior, consider documenting it for users, so it's clear that the `dataList` can't contain `GMX_DATA_ACTION` during shift creation.

It will be wise to add this validation during `createShift` to avoid cancellations. Alternatively, consider passing `new bytes32[](0)` as `dataList` param to the `executeDepositFromController`, just like it's done in the `executeGlvShift`.

Resolution

GMX Team: Resolved.

L-19 | Missing nonReentrant Protection

Category	Severity	Location	Status
Validation	Low	SwapOrderExecutors.sol, IncreaseOrderExecutor.sol, DecreaseOrderExecutor.sol	Resolved

Description

For dependency management, GMX has segregated previous contracts into various executors, such as `IncreaseOrderExecutor`, `SwapOrderExecutor`, and `DecreaseOrderExecutor`. However, these executors currently lack `nonReentrant` protection.

Although we haven't identified a specific exploit path—since the parent contracts (callers) are protected using global or local reentrancy guards—the tradeoff between minimal additional gas and improved safety justifies adding protection at the executor level.

Recommendation

Consider adding `nonReentrant` modifiers to relevant methods within the executor contracts to improve robustness.

Resolution

GMX Team: Resolved.

L-20 | Misspelled Filenames In Executor Contracts

Category	Severity	Location	Status
Best Practices	Low	Global	Resolved

Description

The filenames of both `IncreaseOrderExecutor` and `DecreaseOrderExecutor` are misspelled as `SwapOrdeExecutor` and `DecreaseOrdeExecutor` respectively, with the letter "r" missing in the word "Order".

Recommendation

Consider correcting the filenames to reflect accurate spelling for clarity and maintainability.

Resolution

GMX Team: Resolved.

L-21 | Potential Feed ID Collision Due To Truncation

Category	Severity	Location	Status
Validation	Low	EdgeDataStreamVerifier.sol: 156-157	Acknowledged

Description

In `EdgeDataStreamProvider`, the `leftPadBytes` function truncates the `feedId` to 32 bytes. This can lead to a potential replay issue: if two different `feedId`s share the same first 32 bytes, data for one could be reused or validated incorrectly for the other.

Recommendation

According to Chaos Labs documentation, this scenario appears unlikely given current feed configurations. However, it remains a future risk.

To future-proof the system, especially if you retain control over the signature and signed data format, consider signing the `keccak256` hash of the full `feedId` instead of the raw (or truncated) bytes.

Resolution

GMX Team: Acknowledged.

L-22 | Old Orders Can Overwrite Newer Ones

Category	Severity	Location	Status
Validation	Low	PositionUtils.sol: 803-817	Resolved

Description

There is still the possibility that the user's last `srcChainId` used is not being the one saved in `positionLastSrcChainId`.

Here is an example:

• User opens a position at t0

• User creates a limit order for the position at t1

• At t2 an ADL or liquidation order uses the `srcChainId` from t0 not from t1 even though the data from t1 is more recent

Recommendation

Consider fixing this edge case scenario or document this behaviour.

Resolution

GMX Team: Resolved.

L-23 | Missing Validation For bridgeOutFromController

Category	Severity	Location	Status
Validation	Low	ExecuteDepositUtils.sol: 314	Resolved

Description

During GLV and GM deposits, users will have the option to bridge out these tokens using the `ExecuteDepositUtils.bridgeOutFromController`.

This function will early return if the `dataList` is not correctly configured. However, if `srcChainId = 0`, the GM and GLV tokens were already sent to the receiver, but the transaction will revert as it will try to do a cross chain withdrawal.

Recommendation

Add `srcChainId = 0` to the early return check in `ExecuteDepositUtils.bridgeOutFromController`

Resolution

GMX Team: Resolved.

L-24 | LayerZero Configuration

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

The `MultichainSender` and `MultichainReceiver` are Layer Zero OApps. Therefore, a proper wiring configuration should be performed to guarantee a safe message transferring.

These involve:

- `OApp.setPeer(dstEid, peer)`
- `OApp.setEnforcedOptions()`
- `EndpointV2.setSendLibrary(OApp, dstEid, newLib)`
- `EndpointV2.setReceiveLibrary(OApp, dstEid, newLib, gracePeriod)`
- `EndpointV2.setReceiveLibraryTimeout(OApp, dstEid, lib, gracePeriod)`
- `EndpointV2.setConfig(OApp, sendLibrary, sendConfig)`
- `EndpointV2.setConfig(OApp, receiveLibrary, receiveConfig)`
- `EndpointV2.setDelegate(delegate)`
- Sending chain sending confirmations should match receiving chain receiving confirmations
- Sending chain DVNs should match receiving chain DVNs

Recommendation

Consider adding the proper Layer Zero configuration to these OApps

Resolution

GMX Team: Acknowledged.

L-25 | isSrcChainIdEnabledKey Can Be Bypassed

Category	Severity	Location	Status
Validation	Low	LayerZeroProvider.sol: 172	Resolved

Description

During the `bridgeOut` flow, the user-provided `srcChainId` is validated using the `isSrcChainIdEnabledKey`. However, the actual transfer occurs to the chain identified by `dstEid`, which is resolved via `eidToSrcChainId(cache.dstEid)`.

There is no check to ensure that the user-provided `srcChainId` matches the chain ID derived from `dstEid`. This check can be bypassed by providing a valid `srcChainId` along with an unsupported `dstEid`.

Recommendation

Validate that the provided `srcChainId` matches the chain ID resolved from `dstEid`.

Resolution

GMX Team: Resolved.

L-26 | Keepers May Censor Actions Using BridgeOut

Category	Severity	Location	Status
Censoring	Low	Global	Acknowledged

Description

The Stargate `send` and `sendTokens` functions use a `nonReentrantAndNotPaused` modifier which can be leveraged to force any system's interaction with Stargate to revert.

This can be achieved when the tx originator first enters into the Stargate send function and receives a native fee refund from the `LayerZero` `EndpointV2` contract while within the send function execution.

The malicious tx originator will then call the victim system within the receive function and within the context of the Stargate send function.

Keepers on the GMX platform could leverage this behavior to cause the `StargatePool.send` function to revert, resulting in deposits that are cancelled rather than executed.

Recommendation

Be aware of this censoring vector if GMX keepers are to be decentralized in the future.

Resolution

GMX Team: Acknowledged.

L-27 | Lacking Stargate Fee Protection

Category	Severity	Location	Status
Warning	Low	LayerZeroProvider.sol	Acknowledged

Description

The `minAmount` value for stargate send invocations is determined by the result of the stargate `quoteOft` function. This means the GMX protocol will accept whatever fee the Stargate system reports.

Therefore if users create actions that will `bridgeOut` through stargate and the stargate owner address updates the fee rate to a large value then the user will have no protections against this.

Recommendation

It may be acknowledged that the Stargate owner address is a trusted party, in that case simply be aware of this counterparty risks that GMX users are taking on during the `bridgeOut` flow.

Resolution

GMX Team: Acknowledged.

L-28 | Missing NatSpec Param

Category	Severity	Location	Status
Documentation	Low	MarketPoolValueInfo.sol: 8-18	Resolved

Description

The `lentImpactPoolAmount` parameter has been added to `MarketPoolValueInfo.Props`, but it's missing from the NatSpec-style field comments.

Recommendation

Add `lentImpactPoolAmount` to the struct comments.

Resolution

GMX Team: Resolved.

L-29 | Dangerous Timelock Admin Assignment

Category	Severity	Location	Status
Best Practices	Low	ConfigTimelockController.sol	Acknowledged

Description

In the constructor for the `TimelockController` contract the admin parameter is assigned as the `msg.sender` deploying the `ConfigTimelockController` contract.

This grants unnecessary power to an EOA address, the admin role should be carefully managed. The ideal configuration would be to leave the `TimelockController` as the only address with the `TIMELOCK_ADMIN_ROLE`.

Recommendation

Consider assigning the `admin` address as `address(0)` in the constructor of the `ConfigTimelockController` contract. Otherwise be sure to transfer the `TIMELOCK_ADMIN_ROLE` to a multisig and away from the EOA deploying the system.

Resolution

GMX Team: Acknowledged.

L-30 | Misleading Timelock Transaction Success

Category	Severity	Location	Status
Unexpected Behavior	Low	ConfigTimelockController.sol	Resolved

Description

The `TimelockController` contract from `OpenZeppelin` performs the external call for execution with a simple success check on the result of the call.

This however will incorrectly report the action as successful when attempting to invoke a function on an EOA address.

Recommendation

Be aware of this misleading behavior in the base Open Zeppelin contract and consider overriding `_execute` function if you would like to resolve it.

Resolution

GMX Team: Resolved.

L-31 | Outdated TimelockController Used

Category	Severity	Location	Status
Best Practices	Low	TimelockController.sol	Acknowledged

Description

The `OpenZeppelin` `TimelockController` version used by GMX is outdated. The version used is listed as `last updated v4.9.0`, however recent versions have been updated as recent as `v5.3.0`.

Recommendation

Consider updating the `TimelockController` version to the latest available.

Resolution

GMX Team: Acknowledged.

L-32 | Unfavorable Impact Withdrawal Rounding

Category	Severity	Location	Status
Rounding	Low	MarketPositionImpactPoolUtils.sol	Resolved

Description

The amount withdrawn from the GM markets should be rounded down to favor the protocol market value. However the `longTokenWithdrawalAmount` and `shortTokenWithdrawalAmount` are not always rounded down.

The maximum value for the `longTokenPrice` and `shortTokenPrice` is used. However these values can have much smaller deviations than the `indexTokenPrice` which is also maximized.

In the case where the `indexTokenPrice` has a large spread and the `longTokenPrice` and `shortTokenPrice` have a small spread, the withdrawn amount is actually rounded up rather than down.

Recommendation

Use the minimum of the `indexToken` price when computing the `longTokenWithdrawalAmount` and `shortTokenWithdrawalAmount` to ensure that the amounts withdrawn from the GM market are always minimized.

Resolution

GMX Team: Resolved.

L-33 | Lacking Default Admin Rules

Category	Severity	Location	Status
Best Practices	Low	ConfigTimelockController.sol	Acknowledged

Description

The `ConfigTimelockController` contract inherits from the `TimelockController` contract without implementing the `AccessControlDefaultAdminRules`.

This is typically a recommended safe guard against operations like unintentionally revoking the default admin role.

However the contract adds a significant amount of complexity so it may be best to forgo it's implementation.

Recommendation

Be aware of the lack of safeguards in place when managing the default admin role and interacting with the time lock.

Resolution

GMX Team: Acknowledged.

L-34 | Unnecessary amountLD Param In recordBridgeIn

Category	Severity	Location	Status
Informational	Low	LayerZeroProvider.sol: 118	Resolved

Description

The overloaded `recordTransferIn(token, amount)` function was removed as part of the fixes for L-06 and L-07.

However, the `amountLD` parameter is still being passed to `recordBridgeIn`, even though it is no longer used to determine the bridged amount, which is now fetched directly via the `recordTransferIn(token)` call.

Recommendation

Remove the amount parameter from the `recordBridgeIn` function.

Resolution

GMX Team: Resolved.

L-35 | Inaccurate executionPrice Emitted

Category	Severity	Location	Status
Events	Low	PositionUtils.sol	Acknowledged

Description

In the `getExecutionPriceForDecrease` function only the maximum impact factor is applied to cap the resulting `priceImpactUsd`.

This excludes the capping that is performed based on the contents of the price impact pool and as a result, the `executionPrice` that is computed does not reflect the actual `executionPrice` that is experienced by the order.

Recommendation

Consider re-calculating the `executionPrice` with a version of the `priceImpactUsd` after it has been capped by the impact pool amounts.

Otherwise when computing the `executionPrice` initially, base it off of a version that has been capped by the impact pool amount.

If little complexity should be introduced, consider just acknowledging this and documenting this behavior for consumers of the `executionPrice` emitted on decrease.

Resolution

GMX Team: Acknowledged.

L-36 | Impact Distributions For Pending Impact

Category	Severity	Location	Status
Warning	Low	Global	Acknowledged

Description

The pending impact changes have split the impact pool across three separate variables: the impact pool amount, the pending impact amount, and the `lentAmount`.

However the impact pool amount is still the only variable which experiences decay at the impact pool distribution rate.

With the change to move a significant amount of impact in the market to the pending impact, the magnitude of impact pool distributions will be markedly lowered relative to without it.

Recommendation

Be aware of this change in behavior and consider if it is intended. If it is intended then consider refactoring the configuration of the distribution rate. Otherwise introduce a distribution mechanism for the pending impacts.

Resolution

GMX Team: Acknowledged.

L-37 | maxLendableFactor Configuration

Category	Severity	Location	Status
Configuration	Low	Global	Resolved

Description

The `maxLendableFactor` has been introduced as a sanity check against the price impact available in a market.

However it must now be configured to a non-zero value for every market to ensure positive price impact is not capped to zero, if that is not intended.

Recommendation

Be aware of this configuration requirement and consider if a default value should be introduced.

Resolution

GMX Team: Resolved.

L-38 | Price Impact Griefing

Category	Severity	Location	Status
Warning	Low	Global	Acknowledged

Description

With the introduction of the `maxLendableUsd` validation, it is possible for a malicious LP to intentionally withdraw with an atomic withdrawal to force an account to be unable to realize their positive impact.

Recommendation

This scenario is unlikely to yield any benefit given the LP would likely have to hold a large position and also be exposed to atomic withdrawal fees. However the possibility should be documented for integrations.

Resolution

GMX Team: Acknowledged.

L-39 | Stargate Quote Inaccuracies

Category	Severity	Location	Status
Warning	Low	LayerZeroProvider.sol	Acknowledged

Description

The `quoteOft` function in the `StargatePool` caps the `amountIn` by the available credit in the pool. This behavior creates two unexpected cases for consumers of the `quoteOft` function.

The first case is when the original requested `amountIn` would have produced an `amountOut` that is larger than the credit of the pool. In GMX's case the full `amountIn` is still used in the SendParam.

However the `minAmountLd` is adjusted to be the result of the capped `amountIn` minus fees. This means GMX is assigning a large slippage threshold.

For example:
• Credit available is 10
• Fee rate is 10%
• GMX quotes send for 15
• `amountIn` for the quote is adjusted to 10
• `amountOut` reported is 9
• GMX uses 15 amount in and 9 as the `minAmount`, allowing a fee of 6
• This action reverts anyways as the resulting amount of 13.5 is greater than the available credit

The second case is when the original requested `amountIn` would have produced an `amountOut` that is within the credit of the pool.

For example:
• Credit available is 10
• Fee rate is 10%
• GMX Quotes send for 11
• `amountIn` for the quote is adjusted to 10
• `amountOut` is reported to be 9
• GMX uses the 11 amount and actually receives 9.9 after execution
• The slippage was overallocated in this case

Recommendation

In either case there is no possibility for extraction of the additional slippage room, so no action is necessary. Simply be aware of these edge cases for future use of the `quoteOft` function and any adjustments to the `minAmount`.

Resolution

GMX Team: Acknowledged.

L-40 | Lacking Collateral Factor Validations

Category	Severity	Location	Status
Validation	Low	ConfigUtils.sol	Acknowledged

Description

The general `minCollateralFactor` should always be larger than the `minCollateralFactorForLiquidations` to ensure that users cannot create an immediately liquidatable position however this is not validated in the Config contract.

Recommendation

Be aware of this lacking validation when configuring these variables, otherwise consider implementing the validation in the Config contract.

Resolution

GMX Team: Acknowledged.

L-41 | Asymmetric Capping

Category	Severity	Location	Status
Warning	Low	Global	Acknowledged

Description

As a continuation of the discussion in H-0X and H-0X, this finding serves to call out abnormalities when the negative impact in a market is capped significantly without entirely or nearly entirely reducing positive impact to zero.

Positive impact must be paid for by negative impact as defined by the impact pool, therefore if negative impact is always capped to zero then positive impact cannot be offered.

On the other hand, If a feature is adopted where the lent amount is reserved and cannot be capped away for negative impact, then it may be important to continue to cap the pending positive impact on decrease by the remainder available in the impact amalgam.

This is because, in this paradigm, it is no longer guaranteed that positive impact is realized for a position on increase.

In that sense, the conversion of pending positive impact to a `lentAmount` is what locks in a pending positive impact and users are incentivized to close their position quickly to realize it.

Recommendation

Make clear the intentions for positive impact in the future as this will define several design decisions with the zero impact mechanism.

Resolution

GMX Team: Acknowledged.

L-42 | Functions Cannot Be Used With Multicall

Category	Severity	Location	Status
Warning	Low	MultichainTransferRouter.sol	Acknowledged

Description

The `transferOut` and `bridgeOut` functions in the `MultichainTransferRouter` contract are not marked as `payable` and therefore cannot be used within a multicall context.

Recommendation

Consider if this is the intended behavior or if these functions should be able to be called within a multicall. If they are, then mark these functions as payable.

Resolution

GMX Team: Acknowledged.

L-43 | Missing isSrcChainIdEnabled Check

Category	Severity	Location	Status
Access Control	Low	LayerZeroProvider.sol	Acknowledged

Description

In the `lzCompose` function, if there is no composed deposit action the `isSrcChainIdEnabled` validation is not performed.

Therefore bridges may occur originating from chains which are not enabled with the `isSrcChainIdEnabledKey`.

Recommendation

Consider adding validation on the `isSrcChainIdEnabledKey` for all incoming `lzCompose` actions.

Resolution

GMX Team: Acknowledged.

L-44 | Price Impact Factors Should Be Adjusted

Category	Severity	Location	Status
Warning	Low	Global	Acknowledged

Description

With the introduction of pending price impact, the `totalPriceImpactUsd` that is realized on any given order and position can be multiples of what it was before.

This is notable when comparing against the maximum positive and negative impact factors when these amounts may not be calibrated for the higher magnitude being realized more regularly.

Recommendation

Consider increasing the positive and negative price impact cap factors in accordance with the fact that the magnitude of price impact realized at any one time can be larger. Be sure to also address this in the `isPositionLiquidatable` check.

Resolution

GMX Team: Acknowledged.

L-45 | Redundant WithRelayCache Struct

Category	Severity	Location	Status
Informational	Low	BaseGelatoRelayRouter.sol: 39	Resolved

Description

The `WithRelayCache` struct is no longer used in the modifier after the fix of L-23. However, the struct is still defined in the `BaseGelatoRelayRouter` contract, making it redundant.

Recommendation

Remove the `WithRelayCache` struct.

Resolution

GMX Team: Resolved.

L-46 | Initialize Frontrunning

Category	Severity	Location	Status
Frontrunning	Low	MultichainTransferRouter.sol	Resolved

Description

The `MultichainTransferRouter` contract uses an `initialize` function that is ungated to assign critical values.

This initialize function could be frontrun by a malicious actor during deployment if the initialization is not performed in the same transaction as the deployment.

Recommendation

Consider either adding a trusted modifier to the initialize function or ensuring that the deployment initializes this function in the same transaction.

Resolution

GMX Team: Resolved.

L-47 | Unclear Native Deposit BridgeOut Validation

Category	Severity	Location	Status
Validation	Low	MultichainTransferRouter.sol	Resolved

Description

During the deposit flow, the `bridgeOutFromController` function is invoked even for native deposit actions. For these actions the `srcChainId` will be assigned as 0 and the execution will continue to make an external call to the `multichainTransferRouter` contract `bridgeOutFromController` function.

This function will always revert for deposits that have been made natively on Arbitrum due to the `isSrcChainIdEnabled` validation that occurs in the `_validateCallWithoutSignature` function. The `srcChainId` of 0 will not be enabled.

Recommendation

If it is intended that deposits made on Arbitrum cannot bridge out to other chains, then consider validating this or early returning explicitly and earlier in the execution flow in the `ExecuteDepositUtils.bridgeOutFromController` function.

If it is intended that native deposits should be able to bridge out to other chains, consider refactoring the way this flow is handled such that the `srcChainId` of 0 allows GM or GLV funds to be sent to their target chain.

Resolution

GMX Team: Resolved.

L-48 | Unexpected Deposit Execution Reverts

Category	Severity	Location	Status
Validation	Low	ExecuteDepositUtils.sol: 275	Resolved

Description

Users now can optionally bridge out GM and GLV tokens using Stargate, at the end of the deposit execution.

However, the `bridgeOutFromController` call is not in a try/catch block so any revert in the bridge out flow will make the entire deposit execution fail, cancelling the request.

Users can either inadvertently or maliciously cause the bridge out flow to revert, in different ways:

• Not having enough WNT multichain balance to pay bridge fee or relay fee

• Making an external call fail in `MultichainTransferRouter`

• if `srcChainId` is 0, as it's not an enabled chain

Recommendation

Consider wrapping the `bridgeOutFromController` in a try/catch block to avoid the entire deposit execution to revert. Alternatively, add some validations during deposit creation if the user wants to bridge out the funds.

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