GA GUARDIAN

GMX

GMX Crosschain V2.2 4

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 their GMX Crosschain architecture. From the 9th of June to the 16th of June, 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: b84a69f6bd298695e1bd4ee1e62f593979e83f77

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	0	0	0	0	0	0
High	3	0	0	0	1	2
Medium	15	0	0	9	0	6
Low	18	0	0	9	0	9
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
H-01	Users' Multichain Balance Can Be Consumed	Logical Error	High	Resolved
H-02	Positive Impact Becomes Unbacked	Logical Error	High	Partially Resolved
H-03	Cross-Chain Nonce Handling And Execution Order	Logical Error	High	Resolved
M-01	Max dataList Length Value Too Low	Configuration	Medium	Resolved
M-02	Incorrect positionKey When Updating	Logical Error	Medium	Resolved
M-03	Positive Price Impact Is Not Guaranteed	Logical Error	Medium	Acknowledged
M-04	TraderReferralCode Overwritten	Logical Error	Medium	Acknowledged
M-05	minPositionImpactPoolAmount Change Blocked	Logical Error	Medium	Resolved
M-06	Unset Configuration Keys	Configuration	Medium	Acknowledged
M-07	lzCompose Can Still Be Exploited	Censoring	Medium	Acknowledged
M-08	Some Timelock Actions Cannot Be Done	Unexpected Behaviour	Medium	Resolved
M-09	Max Lent Validation Misses Pool Amount Changes	Validation	Medium	Acknowledged
M-10	Pending Impact Prioritized Over Current Impact	Unexpected Behaviour	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-11	reduceLentAmount Creates Arbitrage	Unexpected Behaviour	Medium	Resolved
M-12	Order Updates Do Not Update lastSrcChainId	Unexpected Behaviour	Medium	Acknowledged
M-13	63/64 Rule Is Not Considered In Validations	Validation	Medium	Acknowledged
M-14	Composed Actions Gas Inaccurately Validated	Censoring	Medium	Acknowledged
M-15	Inaccurate Liquidations Due To Incorrect Capping	Logical Error	Medium	Acknowledged
L-01	Misleading Comment In Constructor	Best Practices	Low	Resolved
L-02	Redundant Logic In LastSrcChainId Update	Best Practices	Low	Resolved
L-03	Cross-chain Actions Cannot Use GMX Pool Liquidity	Warning	Low	Resolved
L-04	Enum Ordering Consistency	Best Practices	Low	Acknowledged
L-05	Newly Added Lending Config Keys Remain Unset	Warning	Low	Acknowledged
L-06	Theoretical Censorship Risk Via Stargate Reentrancy	DoS	Low	Acknowledged
L-07	Signatures Might Be Exposed	Warning	Low	Resolved
L-08	Documentation Regarding executionFee	Documentation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	Bridged Deposit Might Fail Due To Oracle	Warning	Low	Acknowledged
L-10	reduceLentAmount DoS'd With Atomic Swap	DoS	Low	Acknowledged
L-11	reduceLentAmount Rounds Against The Protocol	Rounding	Low	Resolved
L-12	Missing Residual Fee Refunds	Warning	Low	Resolved
L-13	Nonexistent srcChainId Is Not Validated	Validation	Low	Acknowledged
L-14	Duplicated srcChainId Validations	Superfluous Code	Low	Acknowledged
L-15	Price Impact Withdrawal Factor May DoS Users	Configuration	Low	Acknowledged
L-16	Positive Impact Cap Should Early Return	Logical Error	Low	Resolved
L-17	Forcing srcChainId To Equal dstEid Chain	Logical Error	Low	Acknowledged
L-18	TargetIsNotAContract Error Not Used	Superfluous Code	Low	Resolved

H-01 | Users' Multichain Balance Can Be Consumed

Category	Severity	Location	Status
Logical Error	High	MultichainTransferRouter.sol: 86-88	Resolved

Description

`deposit.receiver` is assigned as the account in `bridgeOutFromController` during the `executeDeposit` flow. This account will be responsible for paying the Stargate fee from its multichain balance.

Since anyone can deposit on behalf of another user, the `bridgeOutFromController` function can be exploited to consume any account's `wnt` balance as Stargate fees by donating a minimal amount of `gm` or `glv` tokens to them and invoking a bridge out.

Recommendation

One option to consider is enforcing that `deposit.account` and `deposit.receiver` are the same when calling `bridgeOutFromController`. Another option is to pass both `deposit.account` and `deposit.receiver` to the function.

The `deposit.receiver` should receive the tokens on the source chain, while the Stargate fees should be paid from `deposit.account`'s multichain balance.

Resolution

GMX Team: Resolved.

H-02 | Positive Impact Becomes Unbacked

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol: 924	Partially Resolved

Description

H-04 from the previous round has not been fully resolved. The positive impact is capped based on the impact pool amount using the `capPositiveImpactUsdByPositionImpactPool` function. This cap is applied under the assumption that the entire negative `totalPendingImpactAmount` will be paid without any cap.

However, these pending negative impacts can be capped when decreasing a position. In such cases, the actual amount applied to the position impact pool differs from the delta amount recorded in `totalPendingImpactAmount`.

This means that the `totalImpactPoolAmount` used in the `capPositiveImpactUsdByPositionImpactPool` function is overestimated until these positions with pending negative impact are closed and cap is applied to the pending negative impact. This can be used to gain an advantage by leveraging changes in price impact factors, which occur regularly depending on order book depth.

While the `reduceLentAmount` function can be used to cover unbacked positive impact by paying it from the treasury, overestimation of the positive cap limit persists as long as positions with large pending negative impacts (those that would be capped) remain open.

Recommendation

Since it is not possible to know exactly how much of the negative `totalPendingImpactAmount` will be capped, consider excluding negative pending amounts from positive capping calculations for markets where the negative impact will be capped.

This will result in users receiving less positive impact due to a lower cap, but it will prevent overestimated positive caps and ensure that positive impacts remain fully backed.

Resolution

GMX Team: Partially Resolved.

H-03 | Cross-Chain Nonce Handling And Execution Order

Category	Severity	Location	Status
Logical Error	High	LayerZeroProvider.sol: 91-92	Resolved

Description

As a fix to previously reported issues, GMX now validates cross-chain actions based on signatures. In the lzCompose flow, each call to the GM or GLV multichain router (which increments a user's nonce) is wrapped in a try/catch. When an action fails, its nonce is not updated, and since the failure is caught, LayerZero will not retry it. Any later cross-chain action submitted with old nonce + n will therefore also fail.

Previously this wasn't the case since _validateCallWithoutSignature didn't enforce order, it used to allow action t+1 to go through even if action t failed. While there's no direct financial incentive, a malicious actor could trivially force one failure to deny-of-service all of Alice's pending actions. Consider Alice submitting two cross-chain deposits: Deposit 1 and Deposit 2.

If Deposit 2 is processed before Deposit 1 completes, Deposit 2 will fail (due to a nonce mismatch), caught, and won't be retried and GMX's UI will continue scheduling assuming the next nonce is 3—leading to a desync and permanent failure of future transactions. LayerZero's MessagingComposer.sol does not enforce per-user ordering inside lzCompose, so out-of-order actions are not automatically blocked: link

Considering different chains have different bridging times, if user is scheduling actions from multiple chains, the chances of desync increase. If strict ordering of transactions is enforced, users will be required to wait for their previously triggered cross-chain transactions to fully complete before initiating any source-chain signature-based actions.

Depending on the latency of the source-to-destination chain path, this can block user interactions for a significant duration—potentially 30 minutes or more—resulting in a poor user experience and loss of reactivity during volatile conditions. (which could be considered crucial for perps exchange) For example, if there is incoming crosschain transaction from polygon, until that's executed, user won't be able to sign or use any of their multichain balance on arb/avax.

Recommendation

Consider:
• Option 1: Use a separate nonce queue for cross-chain actions, and validate that the incoming cross-chain nonce equals the current expected nonce before executing any logic (i.e., prior to the try/catch block). This preserves strict ordering within cross-chain flows but decouples it from local nonce progression, allowing local source-chain actions to proceed independently, and since there is explicit reverts, a retry could be attempted.

• Option 2: If strict ordering is not necessary for crosschain actions, adopt a salted nonce model (e.g., combining user nonce with a random salt or action ID). This allows actions to be processed independently without enforcing a sequential queue.

Resolution

GMX Team: Resolved.

M-01 | Max dataList Length Value Too Low

Category	Severity	Location	Status
Configuration	Medium	ControllerUtils.sol: 42	Resolved

Description

The `dataList` parameter will be used to decode GMX action data, like in the `bridgeOutFromController` flow. When creating requests in handler, this parameter is validated against the `MAX_DATA_LENGTH`.

The GMX general config suggest the max length allowed is 10. However, the action data decoded from `dataList` must contain the `relayParams`, which involves multiple params like external calls, oracle params, signature, deadline, etc.

According to our internal testing, even the simplest param encoding (empty external calls, oracle, fee params) will result in a `dataList` length of 41, way above the max configured length.

Recommendation

Consider increasing the `maxDataLength` in the protocol's general config, to allow users to bridge out after deposit.

Resolution

GMX Team: Resolved.

M-02 | Incorrect positionKey When Updating

Category	Severity	Location	Status
Logical Error	Medium	OrderUtils.sol: 213	Resolved

Description

The logic for updating `lastSrcChainId` has been moved from `PositionUtils` to `OrderUtils` after the recent fixes.

Previously, the position itself was used, but now the `positionKey` is derived from the order using `order.initialCollateralToken`.

However, the derived `positionKey` will be incorrect if the `collateralToken` of the position differs from `order.initialCollateralToken` (Similar to issue M-05 from the GMX Crosschain-2 engagement).

As a result, the `lastSrcChainId` of a different position will be incorrectly updated.

Recommendation

To construct the `positionKey` accurately, consider using the actual collateral token, which is the last `tokenOut` in the `swapPath` originating from the `initialCollateralToken`

Resolution

GMX Team: Resolved.

M-03 | Positive Price Impact Is Not Guaranteed

Category	Severity	Location	Status
Logical Error	Medium	Global	Acknowledged

Description

This GMX update introduces pending impact a feature that moves the price impact settlement flow from increase to decrease.

Before this update, the positive price impact in the increase flow was always guaranteed as it was settled immediately.

This is no longer given now for multiple reasons, for example:

• The position impact pool is distributed to the LPs over time

• The amount available for the lending feature is capped

• Prices can change

• During insolvent liquidations the flow might early return before paying the impact pool

Recommendation

Consider acknowledging and documenting this risk for users or rethinking the pending impact implementation, by for example not distributing impact pool funds to the LPs if they are needed to pay out the pending impact.

Resolution

GMX Team: Acknowledged.

M-04 | TraderReferralCode Overwritten

Category	Severity	Location	Status
Logical Error	Medium	LayerZeroProvider.sol: 136	Acknowledged

Description

Usually in the order flow the `TraderReferralCode` is not overwritten if the user already has one set. This check is performed in the `ReferralUtils.setTraderReferralCode` function.

However, in the multichain `setTraderReferralCode` flow the `ReferralStorage.setTraderReferralCode` is called directly instead of through the `ReferralUtils` library, therefore this check is missing.

This means that the actual referrer who onboarded the user to GMX could be overwritten by another one here and fees are lost for the real referrer.

Recommendation

Consider calling `ReferralUtils.setTraderReferralCode` instead of `ReferralStorage.setTraderReferralCode`.

Resolution

GMX Team: Acknowledged.

M-05 | minPositionImpactPoolAmount Change Blocked

Category	Severity	Location	Status
Logical Error	Medium	ConfigUtils.sol: 220-222	Resolved

Description

GMX has added a new check inside `setPositionImpactDistribution` for `minPositionImpactPoolAmount`. It reverts if `minPositionImpactPoolAmount` is less than `totalPendingImpactAmount`: However, these two parameters are not inherently related in core logic.

Consider the following case:

• Pending impact = $500

• Impact pool holds $10,000

• Current `minAmount` = $200

• Admin wants to increase it to $300 or reduce it to $100 — both actions are not possible because of this check.

• But the check isn't necessary in this context, as the impact pool already holds more than enough funds.

Recommendation

Reconsider this check in terms of why it is needed at all.

Resolution

GMX Team: Resolved.

M-06 | Unset Configuration Keys

Category	Severity	Location	Status
Configuration	Medium	LayerZeroProvider.sol: 129	Acknowledged

Description

Several critical configuration keys introduced in recent GMX updates remain unset, potentially leading to execution failures and undefined protocol behavior.

1. Gas Limit Keys – `CREATE_DEPOSIT_GAS_LIMIT`, `CREATE_GLV_DEPOSIT_GAS_LIMIT`
In the current setup, the `lzCompose` function can be called by any actor once DVNs validate the message. If an actor supplies insufficient gas, the subsequent `createDeposit` or `createGlvDeposit` actions may run out of gas and revert to the `catch` block.

While a gas validation step was added in the new implementation, these keys default to `0` unless configured—making the validation ineffective. This could force affected users to retry deposits manually via the routers, degrading UX and increasing vulnerability surface.

2. Lending Constraint Keys – `maxLendableImpactFactorKey`, `maxLendableImpactUsdKey`
These keys are essential for defining safe boundaries for lending impact. Currently, they are not set during deployment. As a result, lending logic may behave unpredictably or allow unsafe exposures due to missing guardrails.

Recommendation

Set `CREATE_DEPOSIT_GAS_LIMIT` and `CREATE_GLV_DEPOSIT_GAS_LIMIT` to appropriate values (e.g., ≥1.4M gas) considering downstream external calls.

Similarly, ensure that `maxLendableImpactFactorKey` and `maxLendableImpactUsdKey` are explicitly configured for each market post-deployment to enforce proper lending constraints.

Resolution

GMX Team: Acknowledged.

M-07 | lzCompose Can Still Be Exploited

Category	Severity	Location	Status
Censoring	Medium	LayerZeroProvider.sol: 91-92	Acknowledged

Description

We previously raised issue M-04 regarding the potential censorship of composed deposits. Even if GMX ensures there is sufficient gas before invoking `try/catch`, censorship remains possible due to the permissionless nature of `lzCompose`.

Any actor can invoke `lzCompose` in a way that ensures the GMX core state causes a revert, effectively censoring the intended action.

For example:

• All GMX actions are protected by a global reentrancy guard. A malicious actor could invoke `lzCompose` during a callback or while receiving ETH for any of their own actions. As a result, if a multichain provider attempts a deposit at this point, it would fail due to the reentrancy check and be caught silently.

• Alternatively, the attacker could sandwich the transaction to trigger a price impact-related revert.

Recommendation

GMX could consider whitelisting specific executors. The `lzCompose` function exposes the executor (i.e., the address that called `lzCompose` on the endpoint). Reference – LayerZero source code

Restricting execution to a trusted set of executors could:
• Mitigate censorship risks.
• Reduce exposure to potential exploits through `lzCompose` calls for any other case.

That said, GMX would need to ensure this whitelist is dynamically managed, as `LayerZero` keepers may change over time. Alternatively, GMX could maintain and operate their own verified set of keepers.

Resolution

GMX Team: Acknowledged.

M-08 | Some Timelock Actions Cannot Be Done

Category	Severity	Location	Status
Unexpected Behaviour	Medium	TimelockConfig.sol	Resolved

Description

In the `TimelockConfig` contract the signal functions are hardcoded to signal a predecessor and salt of 0. This means actions with the same payload will not be able to execute as the id is already used.

Recommendation

For all of the signal functions in the `TimelockConfig` contract, consider allowing for a predecessor and salt to be specified to allow the same payload to be used again in the future and also allow for ordering specific actions.

Otherwise be aware that actions with the same payload cannot be repeated through the `TimelockConfig`.

Resolution

GMX Team: Resolved.

M-09 | Max Lent Validation Misses Pool Amount Changes

Category	Severity	Location	Status
Validation	Medium	DecreasePositionCollateralUtils.sol	Acknowledged

Description

The `capPositiveImpactUsdByPositionImpactPool` function validates that the amount that will become lent is not larger than a percentage of the backing pool amounts.

However this validation is performed on the pool amount prior to the adjustments that are made in the execution of the `processCollateral` function that follows.

As a result the `maxLent` validation can be easily bypassed and allow a malicious actor to lock withdrawals for the GM market or accrue a large index token exposure relative to the backing market token amounts.

Recommendation

Ideally the max lent validation can account for the market balance updates that will occur, however this is non-trivial to implement. Be aware of this shortcoming in the `maxLent` validation and carefully configure the `maxLent` thresholds with this in mind.

Resolution

GMX Team: Acknowledged.

M-10 | Pending Impact Prioritized Over Current Impact

Category	Severity	Location	Status
Unexpected Behaviour	Medium	Global	Resolved

Description

During the decrease price impact flow, the pending price impact amount which has not been capped previously is prioritized over positive impact which is generated on decrease. This occurs with the following calculation in `capPositiveImpactUsdByPositionImpactPool`:

`cache.totalImpactPoolAmount = cache.impactPoolAmount.toInt256() - cache.totalPendingImpactAmount;`

Over time the pending impact is likely to grow larger than the impact pool amount because:

1. The positive impact is no longer capped on increase

2. Negative impact will be capped on decrease, creating an excess of it's positive pending counterpart

This means that over time it is likely that positive impact generated for decrease orders will be un-realizable due to an excess of positive pending impact. This will remove the impact incentive for decrease orders that balance the market.

Recommendation

Be aware of this behavior of the current `capPositiveImpactUsdByPositionImpactPool` and consider if it is expected.

Resolution

GMX Team: Resolved.

M-11 | reduceLentAmount Creates Arbitrage

Category	Severity	Location	Status
Unexpected Behaviour	Medium	PositionImpactPoolUtils.sol	Resolved

Description

The reduceLentAmount function adds the longTokenAmount and shortTokenAmount ratio of the backing tokens to the GM market in return for repaying some of the lentAmount value. The longTokenAmount and shortTokenAmount ratio is paid at the existing ratio of the market token underlying balance.

This however introduces more positive impact that can be realized for markets that are currently in an imbalanced state. Withdrawing tokens from an imbalanced market at the current market ratio is correct, because it reduces the net imbalance of USD values in the market.

However depositing tokens to an imbalanced market at the current market ratio is incorrect, because it increases the net imbalance of USD values in the market.

For example:
• GM A has 1,000 of token A and 10,000 of token B
• The USD imbalance is $9,000
• Withdrawing 50% of the gm supply brings this to $500 and$ 5,000, a USD imbalance of $4,500
• Depositing +50% to the GM supply worth of tokens brings this to $1,500 and$ 15,000, a USD imbalance of $13,500

The USD imbalance is increased for deposits using this mechanism, and thus the pool is more severely off-balance by the price impact calculation measurement.

Recommendation

Keep the existing withdrawal ratio for GM withdrawals and the withdrawFromPositionImpactPool function. However the reduceLentAmount function should deposit new funds into the GM market at an even USD balance to bring the market closer to a balanced state rather than further away.

Resolution

GMX Team: Resolved.

M-12 | Order Updates Do Not Update lastSrcChainId

Category	Severity	Location	Status
Unexpected Behaviour	Medium	Global	Acknowledged

Description

The `lastSrcChainId` is now updated on order creation, however when an order is updated it is not considered for a `lastSrcChainId` update.

A user may create a limit order that is initially never executed due to the trigger price never being reached.

The `lastSrcChainId` of the position may be updated after the inception of the order but before the user updates the order to use a lower trigger price and become executable.

This order update should set a new `lastSrcChainId` since this is the most recent action with the order/position, however it is not considered.

Recommendation

Include `_updatePositionLastSrcChainId` at the end of `updateOrder`.

Resolution

GMX Team: Acknowledged.

M-13 | 63/64 Rule Is Not Considered In Validations

Category	Severity	Location	Status
Validation	Medium	LayerZeroProvider.sol	Acknowledged

Description

In the LayerZeroProvider for the composed actions the _validateGasLeft function is used to validate each of the corresponding gas limits.

However the 63/64 rule is not taken into consideration given that each action is an external call that will only receive 63/64 of the current execution's available gas.

Recommendation

Account for the 63/64 rule in the _validateGasLeft function similar to how it is accounted for in the validateGasLeftForCallback function in CallbackUtils.sol.

Resolution

GMX Team: Acknowledged.

M-14 | Composed Actions Gas Inaccurately Validated

Category	Severity	Location	Status
Censoring	Medium	LayerZeroProvider.sol	Acknowledged

Description

In the LayerZeroProvider the gas required for the execution of composed actions is validated against a static value, however depending on attributes of the composed action the action may require significantly more or less gas.

For example, if an action has several associated permits or external calls then it will consume significantly more gas than the same action without those permits or external calls.

Validating all actions to have the highest expected expenditure would require users to overpay the lz executor, and requiring only the minimum opens these higher gas expenditure actions up to censoring.

Recommendation

Consider estimating the gas required for composed actions based upon additional factors such as the amount of token permits, and external calls associated with the action.

Resolution

GMX Team: Acknowledged.

M-15 | Inaccurate Liquidations Due To Incorrect Capping

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

Description

GMX currently fails to correctly apply caps for negative price impact on `isPositionLiquidatable`, which can incorrectly return `false` when a position should be liquidated.

• 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.

Current market config for `maxPositionImpactFactorForLiquidations` is 0 by default for all markets. Therefore, negative `priceImpactUsd` including pending price impact will be capped at 0.

Recommendation

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.

L-01 | Misleading Comment In Constructor

Category	Severity	Location	Status
Best Practices	Low	MultichainTransferRouter.sol: 18	Resolved

Description

The `MultichainTransferRouter` constructor was previously empty. This allowed users to call `initialize` and set a multichain provider.

This is now prevented be setting a `deployer` address during contract deployment and only allowing this address to call `initialize`.

However the constructor has a comment that state: `leave empty, use initialize instead`, but the function body is not empty. The is misleading and should be removed, as new logic was added.

Recommendation

Remove the comment inside the constructor

Resolution

GMX Team: Resolved.

L-02 | Redundant Logic In LastSrcChainId Update

Category	Severity	Location	Status
Best Practices	Low	OrderUtils.sol: 194-202	Resolved

Description

During the create order flow `order.touch` is called before `_updatePositionLastSrcChainId`. Therefore, the `order.updatedAtTime()` will always equal the current `block.timestamp`.

This makes the whole logic in the `_updatePositionLastSrcChainId` function redundant as at the end it will just update the chain id: `dataStore.setUint(Keys.positionLastSrcChainId(positionKey), srcChainId)`.

Recommendation

Update the `srcChainId` in every order creation by directly calling:

`dataStore.setUint(Keys.positionLastSrcChainId(positionKey), srcChainId);`

Resolution

GMX Team: Resolved.

L-03 | Cross-chain Actions Cannot Use GMX Pool Liquidity

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

The `handleRelayFee` function normally allows users to utilize GMX pool liquidity for performing swaps required to pay fees in the appropriate token.

However, when actions are routed using multichain providers (such as via `lzCompose`), the function returns early.

As a result, users cannot access GMX liquidity and must instead depend on external calls through external swap handlers.

These external handlers cannot be granted the `SwapHandler` controller role because they do not perform internal validations such as `validateSwapPath`, posing potential risks.

Recommendation

Users interacting via cross-chain routing mechanisms like `lzCompose` should be made aware that they will not be able to leverage GMX's native pool liquidity for fee swaps.

A potential enhancement would be to introduce controlled and validated swap paths for such externally routed flows or provide a fallback mechanism via the core contract to retain swap support.

Resolution

GMX Team: Resolved.

L-04 | Enum Ordering Consistency

Category	Severity	Location	Status
Best Practices	Low	IMultichainProvider.sol: 11-12	Acknowledged

Description

A new enum value, `SetTraderReferralCode`, was recently added to the `ActionType` enum used by multichain providers:

```
enum ActionType {
None,
Deposit,
GlvDeposit,
BridgeOut,
SetTraderReferralCode
}
```

Since Solidity enums are nothing but ordered `uint16`s, placing `SetTraderReferralCode` after `BridgeOut` introduces semantic inconsistency.

Recommendation

Consider reordering the enum values for consistency.
`0 to 3: BridgeIn Action`
`4: BridgeOut Action`

```
enum ActionType {
None, //0
Deposit, //1
GlvDeposit, //2
SetTraderReferralCode, //3
BridgeOut //4
}
```

Resolution

GMX Team: Acknowledged.

L-05 | Newly Added Lending Config Keys Remain Unset

Category	Severity	Location	Status
Warning	Low	MarketUtils.sol: 985-986	Acknowledged

Description

GMX has introduced several new configuration keys in the lending logic as part of recent changes. However, these remain unset in the current deployment process.

For instance, the following per-market keys:

• `maxLendableImpactFactorKey`

• `maxLendableImpactUsdKey`

These keys play a important role in determining the boundaries for lending-related impact calculations and risk mitigation, and leaving them unset could lead to undefined or unintended behavior post-deployment.

Recommendation

Be aware that these keys are currently unset and will need to be explicitly configured per market post-deployment to ensure lending behavior aligns with intended risk management.

Resolution

GMX Team: Acknowledged.

L-06 | Theoretical Censorship Risk Via Stargate Reentrancy

Category	Severity	Location	Status
DoS	Low	ExecuteGlvDepositUtils.sol: 134-135	Acknowledged

Description

The `bridgeOutFromController` function allows users to bridge out atomically. However, similar to M-02, this action could theoretically be censored if a malicious actor chooses to interfere.

In this case, censorship could occur through a `nonReentrant-based` revert on the Stargate side. For reference, the `nonReentrant` modifier is enforced here.

A keeper could initiate the call trace from Stargate, invoke a GMX Core action that intentionally causes a revert (e.g., due to reentrancy), and thereby cause the bridgeOut to fail silently.

While there is no rational incentive for keepers to do this under normal assumptions, it remains a theoretical attack vector.

Recommendation

GMX should be aware of this possibility. While action may not be necessary due to the involvement of trusted roles (keepers) who lack any incentive to exploit this vector.

Resolution

GMX Team: Acknowledged.

L-07 | Signatures Might Be Exposed

Category	Severity	Location	Status
Warning	Low	MultichainTransferRouter.sol: 86	Resolved

Description

Users can provide additional data in the `dataList` to invoke the `bridgeOutFromController` flow after deposit execution. This data includes `relayParams`, which contains a signature field.

Normally, a signature is not required for this specific action, as the function validates the call using `_validateCallWithoutSignature`.

Additionally, users cannot know the exact amount that will be bridged out following deposit execution, making it difficult to produce a valid signature.

However, if a user provides a signature thinking it is necessary, the signature will not be used but will still be valid and publicly exposed. Later, anyone could use the same signature to invoke the regular `bridgeOut` flow.

Recommendation

Either ensure that the signature field is empty when using the `bridgeOutFromController` flow, or clearly document this behavior for users.

Resolution

GMX Team: Resolved.

L-08 | Documentation Regarding executionFee

Category	Severity	Location	Status
Documentation	Low	BaseGelatoRelayRouter.sol: 275	Resolved

Description

The `_handleRelayFee` and `_handleRelayAfterAction` functions will return early for calls made through the `LayerZeroProvider` contract, with the addition of the `isRelayFeeExcludedKey`.

Normally, the `_handleRelayFee` function is responsible for handling both the relay fee and the execution fee associated with the deposit.

With the exclusion of `LayerZeroProvider`, the execution fee for cross-chain deposits can no longer be handled within `_handleRelayFee`.

Instead, the execution fee must be transferred to the router contracts within `_handleExternalCalls` during `withRelay`, before the early return in `_handleRelayFee`.

Otherwise, cross-chain deposits will fail due to the router having an insufficient `wnt` balance when attempting to transfer the execution fee to the `depositHandler`.

Recommendation

Document this behavior and inform users about the execution fee handling process.

Resolution

GMX Team: Resolved.

L-09 | Bridged Deposit Might Fail Due To Oracle

Category	Severity	Location	Status
Warning	Low	BaseGelatoRelayRouter.sol: 182	Acknowledged

Description

The `_handleRelayBeforeAction` function has the `withAtomicOraclePrices` modifier. These atomic prices are necessary to be able to perform swaps during the `_handleRelayFee` function. However, `_handleRelayFee` returns early for bridged deposits, so swaps are not performed.

Therefore, setting atomic prices in this case is not necessary. A bridged deposit that would normally succeed might fail during this unnecessary price setting due to issues such as sequencer downtime.

Recommendation

Consider not setting atomic oracle prices if `isRelayFeeExcludedKey` is true, or be aware of this situation.

Resolution

GMX Team: Acknowledged.

L-10 | reduceLentAmount DoS'd With Atomic Swap

Category	Severity	Location	Status
DoS	Low	PositionImpactPoolUtils.sol	Acknowledged

Description

The `reduceLentAmount` function relies on the `fundingAccount` having sufficient approval and balance for the resulting `longTokenAmount` and `shortTokenAmount`.

However these amounts may change before the `reduceLentAmount` function can be executed and cause the execution to fail.

Furthermore a malicious actor may frontrun the `reduceLentAmount` invocation with an atomic swap that intentionally changes the backing ratios of the market such that the `reduceLentAmount` call fails.

In the worst case this could delay the reduction from occurring but can be fixed by ensuring that the `fundingAccount` has sufficient balances for both tokens and sufficient approvals.

Recommendation

Be aware of this DoS vector and plan accordingly.

Resolution

GMX Team: Acknowledged.

L-11 | reduceLentAmount Rounds Against The Protocol

Category	Severity	Location	Status
Rounding	Low	PositionImpactPoolUtils.sol	Resolved

Description

The `reduceLentAmount` function uses the `getProportionalAmounts` function to determine the amounts to be deposited to make up the `reductionAmount`.

The `totalUsd` is maximized using the max price for the index token, however the `getProportionalAmounts` performs round down truncation division on the resulting token amounts as well as the max prices for the total pool value calculation.

Recommendation

Consider adding a boolean to the `getProportionalAmounts` function to indicate that whether the result should be maximized or minimized.

Resolution

GMX Team: Resolved.

L-12 | Missing Residual Fee Refunds

Category	Severity	Location	Status
Warning	Low	BaseGelatoRelayRouter.sol	Resolved

Description

In the `_handleRelayAfterAction` function in the context of a composed action through the multichain provider the execution early returns.

However this misses the `_transferResidualFee` refund which may be necessary if an external call refunded wnt amount to the router contract.

Recommendation

Consider if the `_transferResidualFee` action should still occur for multichain `_handleRelayAfterAction` invocations. If not, be sure to clearly document this for any users or integrations so their wnt is not lost.

Resolution

GMX Team: Resolved.

L-13 | Nonexistent srcChainId Is Not Validated

Category	Severity	Location	Status
Validation	Low	LayerZeroProvider.sol	Acknowledged

Description

In the `_decodeLzComposeMsg` function the `eidToSrcChainId` lookup is used to convert the LZ message's `srcEid` to a `chainId`. However if the src chain is not supported this `eidToSrcChainId` will return an unexpected 0 chain id.

This will simply emit a misleading event and allow the deposit to occur from an unsupported chain. This may also lead to unexpected issues with composed actions.

Recommendation

Consider validating that the resulting `srcChainId` from the `eidToSrcChainId` lookup is not 0 in the `_decodeLzComposeMsg` function.

Resolution

GMX Team: Acknowledged.

L-14 | Duplicated srcChainId Validations

Category	Severity	Location	Status
Superfluous Code	Low	BridgeOutFromControllerUtils.sol: 35	Acknowledged

Description

Both `BridgeOutFromControllerUtils.bridgeOutFromController` and `MultichainTransferRouter.bridgeOutFromController` validate if the `srcChainId = 0` and early return if so.

This was added for a previous issue where native deposits (srcChainId = 0) where trying to bridge out GLV and GM tokens.

Recommendation

Remove one of the duplicated checks

Resolution

GMX Team: Acknowledged.

L-15 | Price Impact Withdrawal Factor May DoS Users

Category	Severity	Location	Status
Configuration	Low	ExecuteWithdrawalUtils.sol: 469	Acknowledged

Description

The `validateMaxLendableFactor` function prevents users from withdrawing funds from the pool if the `lentAmount` is above a certain percentage of the pool amount. This is based on the market's `maxLendableImpactFactorForWithdrawalsKey`.

On the other side, decreasing positions allow users to realize positive price impact up to a percentage of the pool amount, based on the market's `maxLendableImpactFactorKey`, which is different from the withdrawal flow.

Two main issues arise from this logic:

• `validateMaxLendableFactor` is checked on withdrawal execution, not on creation, so users might create un-executable withdrawal orders.

• if `maxLendableImpactFactorForWithdrawalsKey > maxLendableImpactFactorKey`, and a position realized positive price impact using all lendable amount, subsequent withdrawal executions will fail.

In fact, after using the max lendable amount to pay positive price impact, the pool amount will decrease while the `lentAmount` increases. Therefore, `validateMaxLendableFactor` will already fail for withdrawal executions as `lentUsd > maxLendableUsd`

Recommendation

Consider checking for `validateMaxLendableFactor` during withdrawal order creation. Additionally, ensure `maxLendableImpactFactorForWithdrawalsKey` and `maxLendableImpactFactorKey` are correctly configured to avoid withdrawals DoS.

Resolution

GMX Team: Acknowledged.

L-16 | Positive Impact Cap Should Early Return

Category	Severity	Location	Status
Logical Error	Low	MarketUtils.sol: 904	Resolved

Description

The `capPositiveImpactUsdByPositionImpactPool` applies a cap on positive `priceImpactUsd` amounts. However, if the amount is zero, the function will continue, even though it will eventually return zero.

Recommendation

Early return if `priceImpactUsd < 0`.

Resolution

GMX Team: Resolved.

L-17 | Forcing srcChainId To Equal dstEid Chain

Category	Severity	Location	Status
Logical Error	Low	LayerZeroProvider.sol: 182	Acknowledged

Description

The `bridgeOut` flow ensures the `srcChainId` is equal to the chain mapped for `dstEid`. This solves the issue where users could bypass the `isSrcChainIdEnabledKey` and bridge out to a different chain with the `dstEid` param.

However, this forces the user to bridge out to only one specific chain, especially in the following cases:

• bridge in + deposit + `bridgeOut` (`srcChainId` is the chain where funds are bridged from)

• deposit + `bridgeOut` (`srcChainId` is the one user signed the gasless transaction from)

Recommendation

If this is the expected behavior, make sure to document it for users so they are aware of the limitations, and the importance of signing transactions from certain chains when the action will bridge out funds.

Resolution

GMX Team: Acknowledged.

L-18 | TargetIsNotAContract Error Not Used

Category	Severity	Location	Status
Superfluous Code	Low	Errors.sol: 67	Resolved

Description

The `TargetIsNotAContract` is defined in `Errors.sol` contract, but it's never used.

Recommendation

Remove the `TargetIsNotAContract` error.

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