

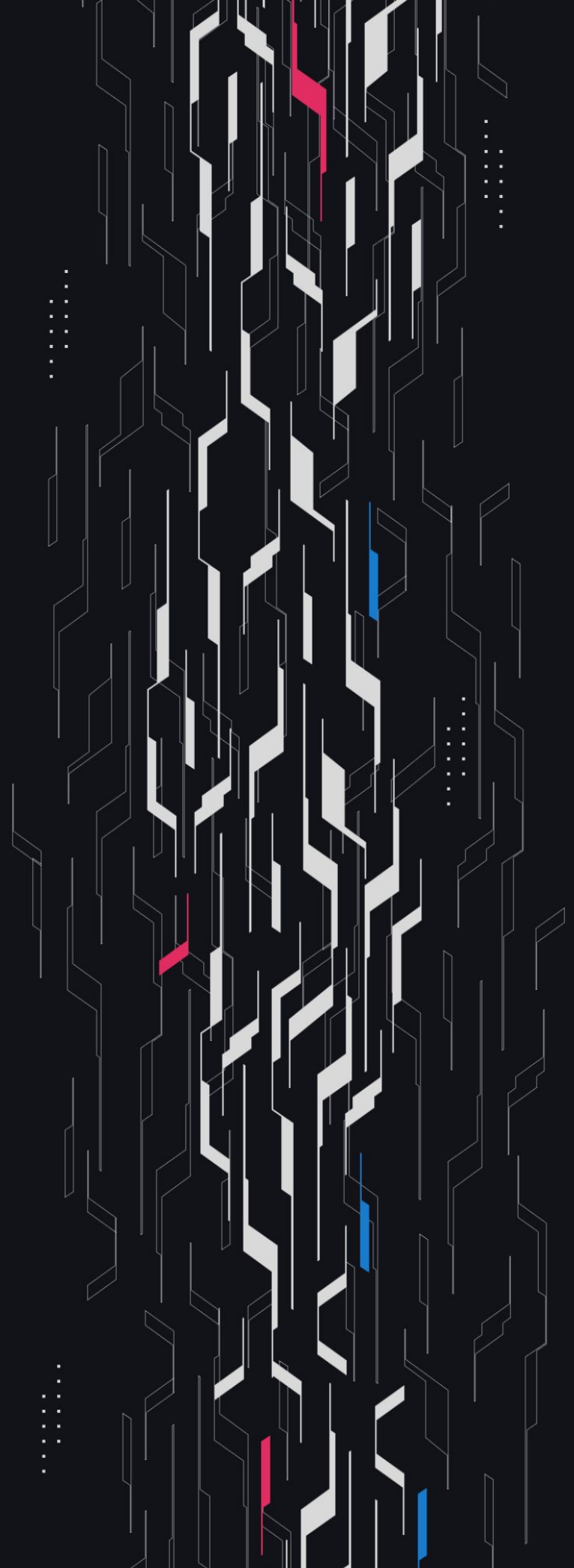
GA GUARDIAN

GMX

**Gasless Sponsored
Calls - 1**

Security Assessment

April 11th, 2025



Summary

Audit Firm Guardian

Prepared By Curiousapple, Wafflemakr, Cosine, Osman Ozedemir, Mark Jonathas

Client Firm GMX

Final Report Date April 11, 2025

Audit Summary

GMX engaged Guardian to review the security of GMX's Gelato Sponsored Call Integration. From the 19 of March to the 21st of March, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

 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-synthetics-team1>,
<https://github.com/GuardianOrg/gmx-synthetics-team2>

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Findings & Resolutions 7

Addendum

Disclaimer 27

About Guardian Audits 28

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics (PR #139 : Gelato sponsored call)
Commit(s)	Initial commit: d3da8d10ab45f7cb5ac87011e7e946c1b627e7bb Final commit: Scheduled for second review for the same scope with remediations and additions

Audit Summary

Delivery Date	April 11, 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	0	0	0	0	0	0
● Medium	1	0	0	0	0	1
● Low	17	0	0	9	1	7

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● 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
M-01	Incorrect Gas Estimation	Logical Error	● Medium	Resolved
L-01	Overestimation Of L1 Poster Fee	Logical Error	● Low	Acknowledged
L-02	Improper Ordering Of oraclePriceCount	Logical Error	● Low	Resolved
L-03	Operational Safeguards	Informational	● Low	Acknowledged
L-04	Trust Assumption: Gelato 1Balance Integration	Trust Assumption	● Low	Partially Resolved
L-05	GMX 1Balance Affected By WETH Price	Logical Error	● Low	Acknowledged
L-06	Relay Params Not Initialized	Configuration	● Low	Resolved
L-07	Insufficient gelatoRelayFeeBaseAmount	Configuration	● Low	Acknowledged
L-08	Incorrect Comment: Calldata Gas Estimation	Informational	● Low	Resolved
L-09	Missing Refund In externalCalls	Logical Error	● Low	Acknowledged
L-10	Swap Fee, Slippage And Bridge Fee Not Taken Into Account	Logical Error	● Low	Resolved
L-11	Missing Market Balance Invariant Check	Validation	● Low	Acknowledged
L-12	Sponsored Call DoS Due To External Dependency	Best Practices	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-13	Partial Calldata Length Used In L2 Fee Estimation	Logical Error	● Low	Acknowledged
L-14	Average Gas Cost Overestimated : calldataLength * 12	Logical Error	● Low	Resolved
L-15	Trust Assumption Subaccounts: Fee Inflation Risks In Sponsored Calls	Trust Assumption	● Low	Resolved
L-16	Trust Assumption Subaccounts: Fee Inflation CallbackGasLimit	Trust Assumption	● Low	Acknowledged
L-17	Trust Assumption Subaccounts: Increase Likelihood Of Arbitrage	Trust Assumption	● Low	Resolved

M-01 | Incorrect Gas Estimation

Category	Severity	Location	Status
Logical Error	● Medium	GasUtils.sol: 588-589	Resolved

Description

GMX estimates the execution fee for sponsored calls using the following formula:

```
// Multiply calldataLength by 2 because the calldata is first sent to the Relay contract, and then to the GMX contract
// Zero byte in calldata costs 4 gas, non-zero byte costs 16 gas; use 12 as a conservative average
uint256 l2Fee = (relayFeeBaseAmount + calldataLength * 2 * 12 + startingGas - gasleft()) * tx.gasprice;
```

Here, `calldataLength * 2 * 12` is used to estimate the calldata cost. The factor of 2 is intended to account for the calldata being sent first to the Relay contract and then to GMX. The average gas cost per byte is assumed to be 12, considering zero and non-zero bytes (4 and 16 gas respectively), which is a reasonable approximation for [transaction creation](#), where this cost is intrinsic and paid upfront before opcode execution.

However, this assumption does not hold for external calls made on-chain (i.e., from the Relay contract to GMX). In external contract calls, calldata is not charged the same way as in transaction creation. Instead, the calldata is copied to memory, and memory expansion costs must be considered. The CALL opcode takes memory offset and size as stack arguments for this reason.

- Key references:
- [EVM Opcodes – CALL](#)
 - [External call gas breakdown](#)
 - [Memory expansion \(quadratic cost\)](#)

Unlike the linear estimation `12 * calldataLength`, memory expansion cost is quadratic in nature. This makes GMX’s fee estimation potentially inaccurate.

- Specifically:
- GMX may underestimate the `l2Fee` when calldata is large (potentially exposing GMX to a loss),
 - And overestimate it when calldata is small, making users repay more than necessary.

For a visual comparison of the gas cost growth curves, refer to this graph: [Desmos graph – Linear vs. Memory Expansion](#)

- For smaller calldata lengths, the linear approximation (`12 * calldataLength`) overestimates the actual memory cost.
- At a certain point, memory expansion begins to drastically outpace the linear estimation.

In theory, an attacker could exploit this by submitting a transaction with a large enough calldata payload, causing the actual memory expansion cost to significantly exceed the estimated fee—leading to GMX covering the shortfall. However, this is unlikely in practice due to the following:

- The intersection point where memory expansion overtakes the linear estimate is around 75 million gas, which is far beyond the current block gas limit. Hence, the worst impact in this case is users paying more to GMX than they should.

Recommendation

Consider removing the "twice" multiplier and approximating the gas cost for memory expansion. Also, include the gas overhead of 700 in the base cost, as that is the base cost for the CALL opcode.

Resolution

GMX Team: Resolved.

L-01 | Overestimation Of L1 Poster Fee

Category	Severity	Location	Status
Logical Error	● Low	GasUtils.sol: 597-598	Acknowledged

Description

GMX derives the L1 poster fee using `Chain.getCurrentTxL1GasFees()`, which fetches the estimated cost of posting the transaction to L1 via the `arbGasInfo` precompile on Arbitrum.

The `relayFeeMultiplierFactor` is meant to account for:

- Potential underestimation of actual gas used (especially on L2),
- And platform-level premiums (e.g., Gelato’s service fee).

Applying this multiplier to L2 fees is reasonable to cover some delta in execution cost thats not accounted in estimation. However, L1 fee estimation is relatively stable.

While it’s not strictly fixed, the value returned by `getCurrentTxL1GasFees()` is usually quite close to the actual cost paid to post the transaction on L1. The delta is small and predictable compared to L2 fee variance.

Currently, GMX applies the same multiplier across both fees:

```
uint256 l1Fee = Chain.getCurrentTxL1GasFees(); uint256 relayFee =
Precision.applyFactor(l1Fee + l2Fee, relayFeeMultiplierFactor);
```

This results in over-scaling the L1 fee, even though it doesn’t need the same buffer, causing users to overpay.

Recommendation

Introduce separate multipliers for L1 and L2 fees. Apply a smaller multiplier to the L1 fee and retain the existing (or configurable) multiplier for L2:

```
uint256 adjustedL1Fee = Precision.applyFactor(l1Fee, l1FeeMultiplierFactor);
uint256 adjustedL2Fee = Precision.applyFactor(l2Fee, l2FeeMultiplierFactor);
uint256 relayFee = adjustedL1Fee + adjustedL2Fee;
```

Resolution

GMX Team: Acknowledged.

L-02 | Improper Ordering Of oraclePriceCount

Category	Severity	Location	Status
Logical Error	● Low	GasUtils.sol: 590	Resolved

Description

While estimating gas in `payGelatoRelayFee`, GMX first accounts for `startingGas - gasleft()` and then performs the `oraclePriceCount` operation.

Recommendation

For more accurate gas estimation, this operation could be performed before accounting for `startingGas - gasleft()`.

Resolution

GMX Team: Resolved.

L-03 | Operational Safeguards

Category	Severity	Location	Status
Informational	● Low	Global	Acknowledged

Description

One of the potential attack vectors when using Gelato's 1Balance feature is that someone could drain GMX's balance by making GMX pay more for relayed transactions than what the user reimburses.

The good part is that the impact of such an attack can be easily capped through a few practical safeguards:

1. Cap 1Balance Allocation with Repeated Top-Ups. Only keep enough funds in the 1Balance account to support a few days of operations. Set up automated or manual top-ups on a rolling basis to avoid leaving large idle balances exposed.

2. Route All UI-Initiated Transactions Through Your Backend. Any gasless transaction initiated from the GMX UI should first pass through your backend, which then relays it to Gelato using your private Gelato API key (keep this key secure at all times).

From your backend, you can:

- Track request frequency per user,
- Apply rate limits,
- And submit only valid, vetted requests to Gelato.

3. Monitor Balance Delta and Auto-Pause on Discrepancy. For every 100 sponsoredCall transactions, monitor the delta between:

- The amount deducted from your 1Balance, and
- The amount reimbursed by users back to your contract.

If there's a persistent discrepancy (indicating underpayment or abuse), you can pause further use of 1Balance from your backend. This is feasible since the API key is owned by you, and all relaying is routed through your infrastructure. A simple background bot/service can be used to automate this monitoring.

4. Set Reasonable Gas Limits and Retry Counts on Gelato. Gelato allows you to configure:

- The maximum gas limit per transaction, and
- The maximum number of retries if a transaction fails.

Set reasonable values for both according to GMX's usecases. You can even configure different max gas limits for different actions since you control the source of all relay actions

5. Whitelisting

We understand that you are already aware of this (as noted in your PR), but it's worth reiterating as a critical operational safeguard: Ensure that only specific addresses and contract methods are whitelisted in Gelato

Recommendation

Consider adding these safeguards to your processes to limit the exposure.

Resolution

GMX Team: Acknowledged.

L-04 | Trust Assumption: Gelato 1Balance Integration

Category	Severity	Location	Status
Trust Assumption	● Low	Global	Partially Resolved

Description

Based on our correspondence with the Gelato team, the following assumptions are made regarding their 1Balance infrastructure and execution pipeline. These are important to note, as deviations from these practices in the future could impact protocol safety:

1. Frontrunning Protection Assumption
 - We assume Gelato uses frontrunning-resistant RPCs during execution to prevent manipulation between simulation and final state execution.
 - This protects against scenarios where an attacker could force a revert post-simulation by altering on-chain state.
2. Secure Simulation Environment Assumption
 - We assume Gelato performs simulations in a forked environment at both the backend and executor levels before broadcasting transactions.
 - These simulations are expected to faithfully reproduce on-chain behavior to avoid unintended execution failures or missed reverts.
 - We also assume there is no significant lag between simulation and execution.
3. Simulation Identity Assumption
 - We assume Gelato does not simulate transactions with tx.origin = address(0) (which is used by GMX to bypass signature checks in their own local simulations).
 - Otherwise, a simulation with incorrect or missing signatures could be considered valid by Gelato, resulting in wasted 1Balance funds.
 - Instead, we assume Gelato uses valid simulation addresses that accurately represent the transaction originator, avoiding bypassable logic paths in client contracts.
4. Fees For Failed Sponsored Calls
 - Gelato currently does not charge its clients for reverted calls, but this behavior may change in the future.

Recommendation

NA

Resolution

GMX Team: Partially Resolved.

L-05 | GMX 1Balance Affected By WETH Price

Category	Severity	Location	Status
Logical Error	● Low	GasUtils.sol: 601	Acknowledged

Description

Relay requests that use `sponsoredCall` will be executed by Gelato, deducting the gas fee from a pre-funded `1Balance` account in USDC (polygon).

However, the user will be charged in WNT during `payGelatoRelayFee`, sent to the `relayFeeAddress`. Consequently, WNT price volatility will directly affect the protocol, as these funds will need to be swapped to USDC and bridged to Polygon chain.

Recommendation

Consider periodically swapping all funds received from GMX, from WNT to USDC to avoid price volatility.

Resolution

GMX Team: Acknowledged.

L-06 | Relay Params Not Initialized

Category	Severity	Location	Status
Configuration	● Low	GasUtils.sol: 569	Resolved

Description

There are 4 new keys added to the GMX config:

- RELAY_FEE_ADDRESS
- RELAY_EXECUTION_GAS_FEE_PER_ORACLE_PRICE
- GELATO_RELAY_FEE_BASE_AMOUNT
- GELATO_RELAY_FEE_MULTIPLIER_FACTOR

However, only the last 2 params are initialised in the deployment script. There are two major impacts:

- all sponsored relay calls will revert, as the relay address is 0
- no additional fee charged for oracles count.

Recommendation

Initialize the 2 missing params to avoid any reverts or undercharging users.

Resolution

GMX Team: Resolved.

L-07 | Insufficient gelatoRelayFeeBaseAmount

Category	Severity	Location	Status
Configuration	● Low	GasUtils.sol: 581	Acknowledged

Description

The current deployment config initializes the following params:

- gelatoRelayFeeMultiplierFactor: 1e30 // 1x
- gelatoRelayFeeBaseAmount: 31000 // 21000 is base gas and GelatoRelay gas

However, there are 2 main issues with this configuration:

- 2 ERC20 token transfers will cost at least 50,000 gas
- Gelato charges a gas premium for each execution (depending on the plan it can range from 8% to 20%), so a 1x factor is very low.
- After gas spent is calculated there is extra gas usage when applying factor, checking relay fee, the after call on the nonReentrant modifier.

These issues will impact directly the fee charged to the user, underestimating the real cost for the transaction.

Recommendation

Consider adjusting the configuration parameters to correctly account for the 2 ERC20 token transfers in the base amount, and apply the same factor Gelato will charge the protocol as gas premium.

Resolution

GMX Team: Acknowledged.

L-08 | Incorrect Comment: Calldata Gas Estimation

Category	Severity	Location	Status
Informational	● Low	GasUtils.sol: 587	Resolved

Description

"zero byte in call data costs 4 bytes, non-zero byte costs 16 bytes" comment in line 587 of GasUtils.sol should be "zero byte in call data costs 4 gas, non-zero byte costs 16 gas".

Recommendation

Consider updating the comment.

Resolution

GMX Team: Resolved.

L-09 | Missing Refund In externalCalls

Category	Severity	Location	Status
Logical Error	● Low	BaseGelatoRelayRouter.sol: 291C1-298C74	Acknowledged

Description

Users have the option to use the `externalHandler` to perform actions outside of GMX, such as interacting with external AMMs to achieve better output amounts.

However, the `_handleRelayFee` logic currently assumes that 100% of the `feeToken` is swapped to WNT (i.e., `swapExactIn`). In scenarios where other swap types are used (e.g., `swapExactOut`), the handler may refund both the `feeToken` and WNT back to the GMX Router.

The issue arises because, after the external call, only the WNT balance is considered as part of the output amount. If a `feeToken` refund occurs during the external call, this refunded amount remains in the `GelatoRouter` contract and is not processed or returned—leading to unintended token retention.

Recommendation

Document this behavior clearly and encourage users to make use of `refundTokens` and `refundReceivers` during external calls to ensure proper fund handling.

Resolution

GMX Team: Acknowledged.

L-10 | Swap Fee, Slippage And Bridge Fee Not Taken Into Account

Category	Severity	Location	Status
Logical Error	● Low	GasUtils.sol: 601	Resolved

Description

The protocol seizes the gas fees during the sponsored call flow in WNT while they need to be paid in USDC. Therefore the GMX itself needs to pay for swapping and bridging fees along with slippage.

Recommendation

Consider adding premium to base cost to account for this delta.

Resolution

GMX Team: Resolved.

L-11 | Missing Market Balance Invariant Check

Category	Severity	Location	Status
Validation	● Low	Global	Acknowledged

Description

The BaseGelatoRelayRouter allows swaps to occur through an arbitrary swap path but does not perform the validateMarketTokenBalance validation on all of the market tokens in the swap path. Same for the removeSubaccount function in the SubaccountGelatoRelayRouter contract.

Recommendation

Implement validateMarketTokenBalance validations.

Resolution

GMX Team: Acknowledged.

L-12 | Sponsored Call DoS Due To External Dependency

Category	Severity	Location	Status
Best Practices	● Low	BaseGelatoRelayRouter.sol: 285	Acknowledged

Description

In the `_handleRelayFee` function, GMX checks whether the caller is the Gelato relay and verifies the `feeToken`. If the caller is the Gelato relay, the fee token must be `WNT`. Otherwise, it can be anything. These values are fetched from calldata using the `GelatoContext` dependency.

```
if (_isGelatoRelay(msg.sender) _getFeeToken() = contracts.wnt) { revert }
```

If the Gelato call is a `syncFee` call, Gelato adds the fee token and some other parameters to the calldata as extra bytes, which are later fetched from calldata. If the call is a sponsored call, no extra bytes are added.

During the check above, it is assumed that sponsored calls are always forwarded from an address other than the Gelato relay since, if the caller were the relay, the call would fail due to the absence of token information.

The issue is that GMX is entirely dependent on the `GelatoContext` dependency. There are two different Gelato relay contracts: `GelatoRelay` and `GelatoRelay1Balance`.

As of now, `GelatoRelay1Balance` is not deployed, not included in the `GelatoContext` dependency, and is not considered a Gelato relay.

However, if `GelatoRelay1Balance` is added to `GelatoContext` as one of the relay contracts after it is deployed, all sponsored calls will fail on the GMX side, as the check in the function assumes that this new Gelato contract will not be considered a relay contract.

Recommendation

Consider implementing internal tracking for Gelato contract addresses, with the option to set them, rather than being dependent on Gelato.

Resolution

GMX Team: Acknowledged.

L-13 | Partial Calldata Length Used In L2 Fee Estimation

Category	Severity	Location	Status
Logical Error	● Low	BaseGelatoRelayRouter.sol: 330-331	Acknowledged

Description

GMX calculates the L2 fee using the following formula:

```
uint256 l2Fee = (relayFeeBaseAmount + calldataLength * 2 * 12 + startingGas - gasleft()) * tx.gasprice;
```

The intention here is to account for the calldata being sent to both the Relay contract and the GMX contract.

However, since the Relay contract performs an external call to the GMX contract, the `msg.data.length` in that context refers only to the calldata passed during the external call, not the original calldata sent to the Relay contract.

With each external call, `msg` properties such as `data`, `sender`, and `value` are reset. The original calldata sent to the Relay contract would look like this:

```
function sponsoredCall (SponsoredCall calldata _call, address _sponsor, address _feeToken, uint256 _oneBalanceChainId, uint256 _nativeToFeeTokenXRateNumerator, uint256 _nativeToFeeTokenXRateDenominator, bytes32 _correlationId) external
```

The calldata length that GMX would actually observe is only the `_call.data` part, which is a subset of the full calldata originally sent to the Relay contract:

```
_call.target.revertingContractCallNoCopy(_call.data, "GelatoRelay.sponsoredCall:");
```

So, using `msg.data.length` during GMX's internal fee calculation only reflects the length of `_call.data`, not the entire input payload originally submitted to the Relay contract.

Recommendation

Since the rest of the parameters in the `sponsoredCall` function (besides `_call.data`) are static in size, their cost could be considered as fixed and accounted for within the `relayFeeBaseAmount` directly, considering incorporating this cost there.

Resolution

GMX Team: Acknowledged.

L-14 | Average Gas Cost Overestimated : calldataLength * 12

Category	Severity	Location	Status
Logical Error	● Low	GasUtils.sol: 592	Resolved

Description

While estimating the L2Fee, GMX multiplies the calldataLength by 12, using it as a conservative average between 4 (for zero bytes) and 16 (for non-zero bytes).

However, this assumption can lead to overcharging users, as evidenced by [this transaction](#) for createOrder.

- The transaction has a calldata size of 1,122 bytes
- Of these, 951 bytes are zero bytes and 171 bytes are non-zero bytes

The actual intrinsic calldata gas cost on-chain was 5,856 gas, but using GMX’s formula:

$1122 * 12 = 13,464 \text{ gas}$

This is more than 2x the actual cost. The overestimation arises because calldata in Solidity is frequently padded with zero bytes to align parameters and structs to 32-byte boundaries. As a result, the average per-byte gas cost is often significantly lower than 12.

Recommendation

Consider looking at onchain GMX's transactions to find the average gas used per byte. Then, add a small buffer to stay safe.

However avoid setting the number too low, because that could underestimate the cost and put GMX’s 1Balance at risk. If unsure, it’s better to keep using 12.

Resolution

GMX Team: Resolved.

L-15 | Trust Assumption Subaccounts: Fee Inflation Risks In Sponsored Calls

Category	Severity	Location	Status
Trust Assumption	● Low	BaseGelatoRelayRouter.sol: 330	Resolved

Description

Any order executed by a sender other than Gelato is classified as a `sponsoredCall`, triggering the `payGelatoRelayFee` function. This calculates the `l2Fee` based on `startingGas`, `calldataLength`, and `oraclePriceCount`.

Since sponsored calls are permissionless, any user or subaccount can execute a pending relay order and append bytes to the `calldata`, inflating `msg.data.length`.

This could increase the relay fee charged to the user, potentially up to their approved fund allowance if a subaccount sets a large `feeAmount`. Unlike `callWithSyncFee`, which was restricted to Gelato initiation, this vulnerability arises due to the open nature of sponsored calls.

In a worst case subaccounts can make users pay more fees than the actual. However, the incentive for such an attack is questionable. A sub account appending data would incur gas costs without gaining any direct benefit, as the increased fee is paid to GMX, not the sub account.

Additionally, GMX already accepts more significant risks from malicious sub account behavior, suggesting this issue may not be a primary concern.

Recommendation

Consider limiting the `msg.data.length` for sponsored calls to prevent potential fee inflation

Resolution

GMX Team: Resolved.

L-16 | Trust Assumption Subaccounts: Fee Inflation CallbackGasLimit

Category	Severity	Location	Status
Trust Assumption	● Low	SubaccountRouter.sol	Acknowledged

Description

The GMX team has previously acknowledged the risk of a slow fund bleed up to a maximum count as a valid trust assumption. However, uncertainty about their awareness of the `callbackGasLimit` factor prompted us to raise this issue again.

A malicious subaccount can exploit the main account by:

- Creating an un-executable limit order (e.g., with an unrealistic trigger price), specifying a callback contract and maximum execution fee.
- Canceling the order to claim the excess fee via the callback contract, then repeating the process.

The protocol mitigates this with features like `setMaxAllowedSubaccountActionCount` and execution fee caps for sub accounts.

However, the cap relies on an arbitrary `callbackGasLimit` (within a maximum) set by the subaccount, allowing a compromised sub account to still siphon funds—albeit in reduced amounts—despite transaction limits.

Recommendation

- Document this residual risk in your sub account trust assumptions
- OR
- Prohibit sub accounts from using callbacks in orders to eliminate the vulnerability.

Resolution

GMX Team: Acknowledged.

L-17 | Trust Assumption Subaccounts: Increase Likelihood Of Arbitrage

Category	Severity	Location	Status
Trust Assumption	● Low	RelayUtils.sol: 43	Resolved

Description

Users can convert a relay fee token to WNT via GMX atomic swaps in gasless orders. These swaps lack slippage protection, but the `outputAmount` should cover relay and execution fees.

Unlike the permissioned `callWithSyncFee`, restricted to Gelato executors, the new `sponsoredCall` feature allows anyone to process gasless orders.

This enables a sub account to create a gasless order with an atomic swap and inflated `feeAmount`, then execute it at a specific time to induce a large negative price impact. The sub account could arbitrage this with a personal order to capture the positive price impact.

Though router approval via token permit is required, the swapped token may already be the collateral token, not limiting this to relay fees. GMX documentation notes a related risk: “a malicious user could create trades with high price impact in an attempt to profit.”

However, the combination of atomic swaps and permissionless `sponsoredCall` makes this exploit deterministic, not merely an attempt, increasing its likelihood.

Recommendation

Consider adding price impact protection for fee swaps and GMX orders to counter sub account exploitation, or acknowledge the heightened risk due to deterministic incentives.

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 Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits 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>