GUARDIAN

GMX

GMX Fee Automations

Security Assessment

October 29th, 2025

Summary

Audit Firm Guardian

Prepared By Curiousapple, Cosine

Client Firm GMX

Final Report Date October 29, 2025

Audit Summary

GMX engaged Guardian to review the security of their GMX Fee Automations. From the 19th of August to the 26th of August, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian considers the codebase suitable for deployment. Guardian advises the protocol to consider periodic review with future changes.

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

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics
Commit(s)	Initial commit: 8d920064b47c83b020ccf301b7df507832380b57 Final commit: 7fde9c58a110ff2a99efd528af95006f7448cb8c

Audit Summary

Delivery Date	October 29, 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	1	0	0
● Low	3	0	0	1	0	2
● Info	10	0	0	6	0	4

Audit Scope & Methodology

Scope and details:

https://github.com/gmx-io/gmx-synthetics/pull/94

contracts/claim/ClaimHandler.sol
contracts/claim/ClaimUtils.sol
contracts/config/Config.sol
contracts/config/TimelockConfig.sol
contracts/contributor/ContributorHandler.sol
contracts/data/Keys.sol
contracts/data/Keys2.sol
contracts/error/Errors.sol
contracts/event/EventUtils.sol
contracts/fee/FeeDistributor.sol
contracts/fee/FeeDistributorUtils.sol
contracts/fee/FeeDistributorVault.sol
contracts/fee/FeeHandler.sol
contracts/multichain/MultichainReader.sol
contracts/multichain/MultichainReaderUtils.sol
contracts/role/Role.sol
contracts/role/RoleModule.sol

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
M-01	Keepers Can Steal Fees	Logical Error	● Medium	Acknowledged
L-01	FeeDistributor Can Permanently Stuck	Logical Error	● Low	Acknowledged
L-02	Missing Token Validation	Validation	● Low	Resolved
L-03	Missing Action In Timelock Config	Logical Error	● Low	Resolved
I-01	Redundant Check In initiateDistribute	Superfluous Code	● Info	Acknowledged
I-02	Invalid NatSpec	Superfluous Code	● Info	Resolved
I-03	Treasury Funds Risk Exposure	Warning	● Info	Resolved
I-04	Blacklisted Token Transfers	Warning	● Info	Acknowledged
I-05	Static Absolute Contributor Amounts	Configuration	● Info	Acknowledged
I-06	Missing LayerZero Read Channels	Configuration	● Info	Acknowledged
I-07	On-Chain Test Runs Required	Best Practices	● Info	Acknowledged
I-08	Fee Distribution Period Overlap	Warning	● Info	Resolved
I-09	Fee Distribution Requires Synchronous Execution	Warning	● Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-10	Bridging Excess GMX Requires Native Fee	Warning	● Info	Acknowledged

M-01 | Keepers Can Steal Fees

Category	Severity	Location	Status
Logical Error	● Medium	FeeDistributorUtils.sol: 36-37	Acknowledged

Description

Since the keeper's raw ETH balance is used for comparison against the target, they can move funds to claim more than what is actually attributed to them.

Additionally, because keepers are responsible for performing certain on-chain actions, they always need to maintain some funds to run those operations.

The calculation target – current balance fails to account for the operating balance required for those actions.

Recommendation

Consider sending the amounts directly in `initiateDistribution` instead of relying on the keeper's balance, or be aware that in this case, keepers could theoretically always claim up to `FEE_DISTRIBUTOR_KEEPER_COSTS`

Resolution

GMX Team: Acknowledged.

L-01 | FeeDistributor Can Permanently Stuck

Category	Severity	Location	Status
Logical Error	● Low	Global	Acknowledged

Description

The FeeDistributor contract uses validateReadResponseTimestamp to enforce that LayerZero (LZ) read responses are received within the configured maximum delay (FEE_DISTRIBUTOR_MAX_READ_RESPONSE_DELAY).

Similarly, Gelato keeper calls are expected within this time frame.

If a response or keeper call arrives late, the corresponding method (e.g. processLzReceive) will revert, leaving the contract stuck in the Initiated state. In this state:

• initiateDistribute cannot be called again (requires state None).

• No other function transitions state back to None.

• As a result, the distribution process cannot be restarted.

This creates a deadlock scenario where a single delayed or missing read/keeper execution can halt fee distributions indefinitely.

Weekly fee distribution may become permanently blocked if one LZ read or Gelato execution is delayed.

Recovery is impossible without a contract upgrade or manual intervention.

Recommendation

Consider allowing a reset of the state back to None if the system has been in Initiated (or ReadDataReceived) longer than FEE_DISTRIBUTOR_MAX_READ_RESPONSE_DELAY.

This could be implemented as a restricted function callable by the fee distribution keeper or governance.

Resolution

GMX Team: Acknowledged.

L-02 | Missing Token Validation

Category	Severity	Location	Status
Validation	● Low	ContributorHandler.sol: 76-77	Resolved

Description

The function setMaxTotalContributorTokenAmount does not verify whether the provided token exists in the DataStore.

As a result, an arbitrary token can be passed and a max value set, effectively bypassing the intended max validation logic. _validateMaxContributorTokenAmounts

This contrasts with setContributorAmount, where the token existence check is correctly enforced — suggesting the validation was unintentionally omitted in setMaxTotalContributorTokenAmount.

Recommendation

Add a token existence check in setMaxTotalContributorTokenAmount.

Resolution

GMX Team: The issue was resolved in ContributorHandler.sol#L87.

L-03 | Missing Action In Timelock Config

Category	Severity	Location	Status
Logical Error	● Low	ContributorHandler.sol: 76-77	Resolved

Description

The function `setMaxTotalContributorTokenAmount` can only be called by the controller, but it does not have a corresponding `set` action in the `TimelockConfig`.

Hence, this parameter is not configurable.

A similar concern applies to:

- `setMinContributorPaymentInterval`

Recommendation

Consider adding corresponding timelock functions

Resolution

GMX Team: The issue was resolved in TimelockConfig.sol#L477.

I-01 | Redundant Check In initiateDistribute

Category	Severity	Location	Status
Superfluous Code	● Info	FeeDistributor.sol: 75-79	Acknowledged

Description

The initiateDistribute function checks that getAddress(Keys.FEE_RECEIVER) equals address(feeDistributorVault) and reverts otherwise.

Checking a configured variable against another configured variable is a redundant operation.

Recommendation

Consider saving the address only once and always use the same.

Resolution

GMX Team: Acknowledged.

I-02 | Invalid NatSpec

Category	Severity	Location	Status
Superfluous Code	● Info	FeeHandler.sol: 52	Resolved

Description

The NatSpec of the withdrawFees function talks about a marketTokens parameter that does not exist.

Recommendation

Consider removing that comment.

Resolution

GMX Team: The issue was resolved in FeeHandler.sol#L52.

I-03 | Treasury Funds Risk Exposure

Category	Severity	Location	Status
Warning	● Info	Global	Resolved

Description

In discussion, the GMX team indicated that the vault address in ContributorHandler could potentially be the DAO treasury, a multisig, or one of the v1 timelocks.

If the vault is set to the DAO treasury (or any address holding more funds than those intended for contributor distribution), all of those funds would effectively be exposed to the contributor keeper, controller, and contributor distributor.

Since keepers are expected to be managed by automated scripts, this greatly increases the risk surface.

A similar pattern exists in the FeeDistributor:

• In finalizeWntForTreasury (L557–L558), the GMX treasury would need to approve balances to the FeeDistributor.

Thus, both components share the same underlying problem: core treasury funds are unnecessarily exposed through distribution mechanisms.

Recommendation

• Explicitly restrict the vault to a dedicated distribution contract or dedicated multisig, which should only hold the funds required for periodic distributions.

• Avoid using the DAO treasury (or any address with significant unrelated balances) directly in contributor or fee distribution

Resolution

GMX Team: The issue was resolved in Keys.sol#L474.

I-04 | Blacklisted Token Transfers

Category	Severity	Location	Status
Warning	● Info	ContributorHandler.sol: 113-114	Acknowledged

Description

Some ERC20 tokens revert on zero-balance or blacklisted transfers, meaning a single problematic contributor could cause all payments for all tokens and contributors to fail, though this risk is mitigated as contributor keeper can remove such contributors.

Recommendation

Beware of this scenario.

Resolution

GMX Team: Acknowledged.

I-05 | Static Absolute Contributor Amounts

Category	Severity	Location	Status
Configuration	● Info	ContributorHandler.sol	Acknowledged

Description

In ContributorHandler, contributor payments are currently set using absolute token amounts. This design is rigid and introduces problems under varying vault balances:

If the vault balance is less than the total contributor amounts, sendPayments would revert.

If the vault balance is greater than the total contributor amounts, the surplus remains unallocated.

Since setContributorAmount applies immediately (pre-action) and persists post-action, it would be applicable for time spent from last payment to present.

Recommendation

Execute the flow below for the current code:

1. Calculate contributor amounts based on what you intend to distribute in the upcoming payment.

2. Call setContributorAmount for all contributors (multicall is fine).

3. Call sendPayments.

4. (Optional but recommended) Reset all contributor amounts to zero afterward.

Resolution

GMX Team: Acknowledged.

I-06 | Missing LayerZero Read Channels

Category	Severity	Location	Status
Configuration	● Info	Global	Acknowledged

Description

As per our communication with GMX, they have future plans for using Botonix alongside other deployments for fee calculation and distribution.

However, review of the LayerZero deployments shows that Botonix does not have read channels enabled.

In addition, Botonix is not listed as a target data chain for either AVAX or ARB under any DVN provider.

As a result, cross-chain read communication between these networks is currently not supported:

• ARB/AVAX cannot read data from Botonix.

• Botonix cannot read data from ARB/AVAX.

This limitation means that critical reads, such as fetching total staked or supply values, cannot be performed under current LayerZero read channel support.

Recommendation

Beware that Botonix state can’t be used for this automatic fee distribution until read channels are supported there.

Resolution

GMX Team: Acknowledged.

I-07 | On-Chain Test Runs Required

Category	Severity	Location	Status
Best Practices	● Info	Global	Acknowledged

Description

Given the involvement of third-party integrations with opaque code, such as LayerZero DVNs, it is critical that the flow is first tested on testnet (or on mainnet with trial configs and restricted access).

Certain configurations can only be reliably validated through on-chain traces, making this step essential.

Recommendation

• Conduct a trial run of fee distribution on testnet/mainnet for both AVAX and ARB.

• Share the resulting on-chain traces with us for review and verification of correctness.

• Pay particular attention to gas limits for LayerZero actions, as these are best validated directly on-chain.

Notes For Guardian Team During Remediations

Check returnDataSize behaviour onchain, +8 requirement

Check bridgeGMX flow, when layerzeroOFT = gmx

Resolution

GMX Team: Acknowledged.

I-08 | Fee Distribution Period Overlap

Category	Severity	Location	Status
Warning	● Info	FeeDistributor.sol: 267-268	Resolved

Description

The current check in FeeDistributor.sol only verifies that the calculated amount meets the minimum bridging requirement:

```
// if the calculated amount doesn't meet the min bridging requirement, revert
if (feeAmountGmxCurrentChain < minRequiredGmxAmount) {
revert Errors.BridgedAmountNotSufficient(minRequiredGmxAmount, feeAmountGmxCurrentChain);
}
```

However, this check does not account for scenarios involving overlapping fee distribution periods:
• Suppose there are two periods: x → y and y → z.
• Distribution for x → y is initiated but insufficient funds are available on-chain.
• While waiting for bridged GMX to arrive, the system progresses into y → z, during which new fees may already have been deposited into the vault.

This can lead to situations where:

`vault balance > fundsOnChain + fundsBridged`

because the vault balance now includes funds from a future period.

While not a critical vulnerability, it could cause confusing or inconsistent assertions when reviewing balances and tracing distributions.

Recommendation

• Be aware of this behavior when designing assertions and flows.
• Alternatively, document this limitation clearly, as it may not be a significant issue but could affect operational clarity.

Resolution

GMX Team: The issue was resolved in FeeDistributor.sol#L75.

I-09 | Fee Distribution Requires Synchronous Execution

Category	Severity	Location	Status
Warning	● Info	Global	Resolved

Description

The current fee distribution design assumes that all steps are executed synchronously across all chains for a given state.

This reliance on synchronous execution across various chains introduces operational risk if one chain or keeper on one chain experiences downtime

Recommendation

• Always execute fee distribution actions in sync across all chains.

• Establish a contingency plan for scenarios where one chain (e.g., Arbitrum) experiences downtime or delayed execution.

• Share this contingency plan with us for review to ensure consistency and operational safety.

Resolution

GMX Team: The issue was resolved in FeeDistributor.sol#L75.

I-10 | Bridging Excess GMX Requires Native Fee

Category	Severity	Location	Status
Warning	● Info	FeeDistributor.sol: 479-480	Acknowledged

Description

In FeeDistributor.sol (L479–L480), bridging excess GMX requires a native fee to be paid by the contract itself.

This means the contract must always hold some native ETH balance (or the relevant native token on other chains) in order to successfully perform cross-chain sends.

Recommendation

Ensure that the contract maintains a minimum buffer of native ETH to cover bridging fees.

Resolution

GMX Team: Acknowledged.

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