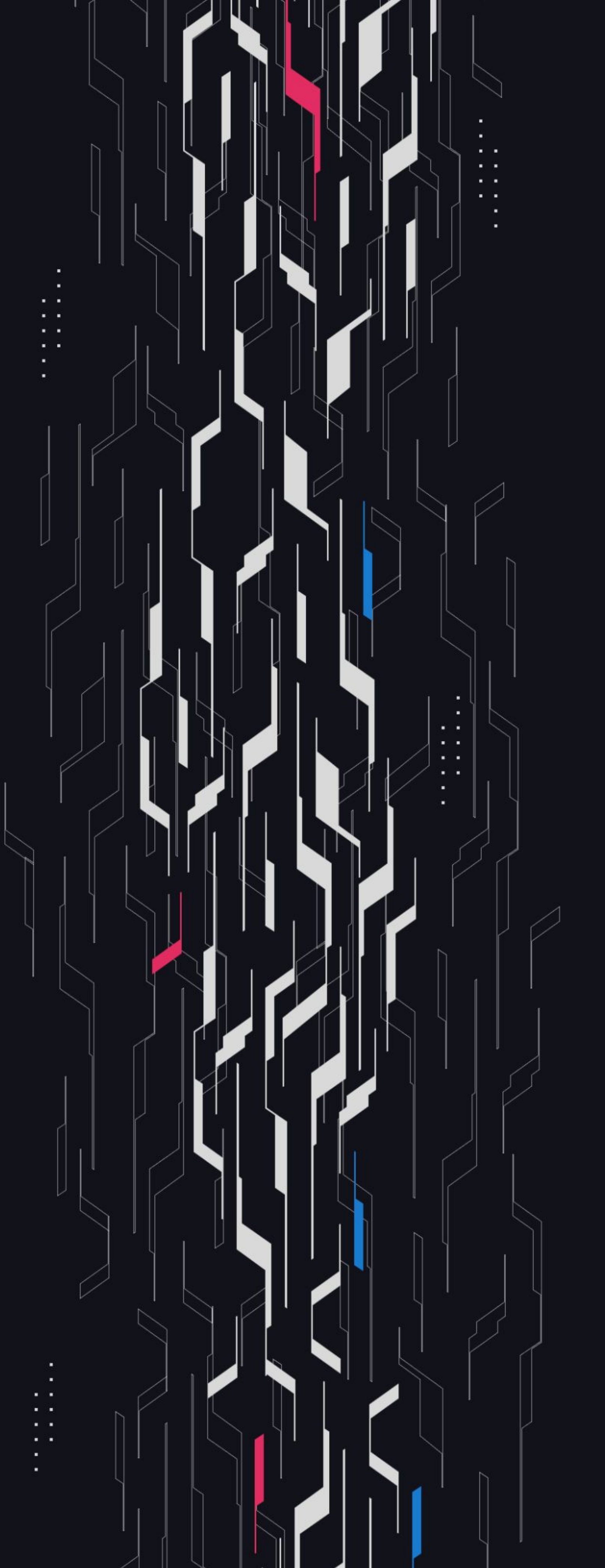
GUARDIAN

GMX

V2.1 Updates

Security Assessment

June 14th, 2024



Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Kiki, Mark Jonathas, Vladimir Zotov, Dogus Kose, 0x3b

Client Firm GMX

Final Report Date June 14th, 2024

Audit Summary

GMX engaged Guardian to review the security of updates to it's synthetic assets exchange. From the 6th of May to the 27th of May, a team of 7 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/GuardianAudits/gmx-v2-1-fuzzing

Table of Contents

Project Information

Smart Contract Risk Assessment

Addendum

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics/
Commit(s)	711ceb2f5e278afdd290da14794dec3427e4c525

Audit Summary

Delivery Date	June 14th, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	3	0	0	0	0	3
● High	9	0	0	1	1	7
● Medium	9	0	0	6	0	3
● Low	24	0	0	13	0	11

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.

Invariants Assessed

During Guardian's review of GMX V2.1, fuzz-testing with Echidna was performed on the protocol's main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Passed	Remediation	Run Count
INC-01	Position size in USD should increase after successful increase position call.	✅	✅	10,000,000+
INC-02	Long Open Interest should increase after successful increase position call.	✅	✅	10,000,000+
INC-03	Collateral amount of position should increase after successful increase position call.	✅	✅	10,000,000+
INC-04	Collateral sum for longs should increase after successful increase position call.	✅	✅	10,000,000+
DEC-01	Position size in USD should decrease after successful decrease position call.	✅	✅	10,000,000+
INC-02	Long Open Interest should increase after successful increase position call.	✅	✅	10,000,000+
INC-03	Collateral amount of position should increase after successful increase position call.	✅	✅	10,000,000+
INC-04	Collateral sum for longs should increase after successful increase position call.	✅	✅	10,000,000+
DEC-01	Position size in USD should decrease after successful decrease position call.	✅	✅	10,000,000+
DEC-02	Collateral amount of position should decrease after successful decrease position call.	✅	✅	10,000,000+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
DEC-03	Long Open Interest should decrease after successful decrease position call.	✅	✅	10,000,000+
DEC-04	Collateral sum for longs should decrease after successful decrease position call.	✅	✅	10,000,000+
CLOSE-01	Position size in USD should be 0 after closing the position.	✅	✅	10,000,000+
CLOSE-02	Position size in tokens should be 0 after closing the position.	✅	✅	10,000,000+
CLOSE-03	Position collateral amount should be 0 after closing the position.	✅	✅	10,000,000+
CLOSE-04	Auto cancel order list should be empty after closing the position.	✅	✅	10,000,000+
DEP-1	Deposited market token amount should be equal to the amount after simulation.	✅	✅	10,000,000+
DEP-2	Market tokens total supply should increase after deposit.	✅	✅	10,000,000+
WITHD-1	The user should not be able to withdraw any long tokens with 0 LP tokens.	✅	✅	10,000,000+
WITHD-2	The user should not be able to withdraw any short tokens with 0 LP tokens.	✅	✅	10,000,000+
WITHD-3	Withdrawn long token amount should be equal to amount after simulation.	✅	✅	10,000,000+
WITHD-4	Withdrawn short token amount should be equal to amount after simulation.	✅	✅	10,000,000+
WITHD-5	Market tokens total supply should decrease after withdrawal.	✅	✅	10,000,000+
SHFT-1	User balance of from GM tokens decreases upon shift.	✅	✅	10,000,000+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
SHFT-2	User balance of to market GM increases by shift.marketTokenAmount().	✅	✅	10,000,000+
SHFT-3	User balance of from market GM decreases by shift.marketTokenAmount().	✅	✅	10,000,000+
SHFT-4	Claimable fees for long token do not change upon shift.	✅	✅	10,000,000+
SHFT-5	Claimable fees for short token do not change upon shift.	✅	✅	10,000,000+
SHFT-6	Long token pool amount for from market should decrease if simulateLongTokenAmountWithdrawal > 0.	✅	✅	10,000,000+
SHFT-7	Short token pool amount for from market should decrease if simulateShortTokenAmountWithdrawal > 0.	✅	✅	10,000,000+
SHFT-8	Pool amount for to market long token should increase if simulateLongTokenAmountWithdrawal > 0.	✅	✅	10,000,000+
SHFT-9	Pool amount for to market short token should increase if simulateShortTokenAmountWithdrawal > 0.	✅	✅	10,000,000+
SHFT-10	Market token (GM) value for from market stays the same after shift execution.	✅	✅	10,000,000+
SHFT-11	Market token (GM) value for to market stays the same after shift execution.	✅	✅	10,000,000+
LIQ-01	Position count should decrease after liquidation.	N/A	✅	1,000,000+
LIQ-02	Auto cancel order list should be empty after liquidation.	N/A	✅	1,000,000+
ADL-01	Position size should be reduced exactly by delta	N/A	✅	1,000,000+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
CNCL-DEP-01	User market token amounts should stay unchanged	N/A	✅	1,000,000+
CNCL-DEP-02	User long token amounts after cancel should be less or equal balance before plus deposited amount	N/A	✅	1,000,000+
CNCL-DEP-03	User long token amounts should stay unchanged (stronger invariant than CNCL-DEP-02)	N/A	✅	1,000,000+
CNCL-DEP-04	Vault long token amounts should stay unchanged	N/A	✅	1,000,000+
CNCL-DEP-05	User short token amounts after cancel should be less or equal balance before plus deposited amount	N/A	✅	1,000,000+
CNCL-DEP-06	User short token amounts should stay unchanged (stronger invariant than CNCL-DEP-05)	N/A	✅	1,000,000+
CNCL-DEP-07	Vault short token amounts should stay unchanged	N/A	✅	1,000,000+
CNCL-ORD-1	User should receive the same amount of long tokens he sent to create an order	N/A	✅	1,000,000+
CNCL-ORD-2	User should receive the same amount of short tokens he sent to create an order	N/A	✅	1,000,000+
CNCL-SHFT-01	Market from received amount should be less or equal than before	N/A	✅	1,000,000+
CNCL-SHFT-02	Market to balance should stay unchanged	N/A	✅	1,000,000+
CNCL-WITH-1	User should receive market tokens back after cancelling withdrawal	N/A	✅	1,000,000+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
CNCL-WITH-2	Market tokens total supply should stay the same	N/A	✅	1,000,000+
CNCL-WITH-3	Vault should refund market tokens	N/A	✅	1,000,000+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	External Call Gas Adjustment DoS's Liquidations	Logical Error	● Critical	Resolved
C-02	Cancel Callbacks Prevent Liquidations	Logical Error	● Critical	Resolved
C-03	shiftGasLimitKey Returns Incorrect Gas Limit Key	Logical Error	● Critical	Resolved
H-01	Borrowing Fees Increase Based On Incorrect Rate	Logical Error	● High	Resolved
H-02	Atomic Providers Cannot Be Configured	Logical Error	● High	Resolved
H-03	Users Can Use Shift To Avoid Deposit Fees	Logical Error	● High	Resolved
H-04	Cannot Set Data Stream Through Config	Validation	● High	Resolved
H-05	GMOracleProvider Reverts Due To Incorrect Validation	Logical Error	● High	Resolved
H-06	Gas Validation Does Not Account For Callback Gas	Logical Error	● High	Resolved
H-07	MarketSwap Orders May Use Unexpected Prices	Validation	● High	Partially Resolved
H-08	Atomic Withdrawal Feature Unusable	Logical Error	● High	Resolved
H-09	Migration Inconsistencies	Logical Error	● High	Acknowledged
M-01	makeExternalCalls Unexpected Funds Receiver	Logical Error	● Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-02	Attack can game price impact at users expense	Logical Error	● Medium	Acknowledged
M-03	MAX_AUTO_CANCEL_ORDERS Update Risk	Unexpected Behavior	● Medium	Acknowledged
M-04	uiFee is Taken Twice During Shift	Logical Error	● Medium	Resolved
M-05	Dangerous Atomic Withdrawal Invocation Pattern	Unexpected Behavior	● Medium	Resolved
M-06	Shifts Within A Virtual Inventory Unfairly Punished	Unexpected Behavior	● Medium	Resolved
M-07	setFundingRate Unexpectedly Changes Funding Fees	Unexpected Behavior	● Medium	Acknowledged
M-08	MEV Bribes Allow Griefing Of Keepers	Protocol Manipulation	● Medium	Acknowledged
M-09	Refund gas limit is not accounted for	Validation	● Medium	Acknowledged
L-01	Kink borrowing rate charges base and optimal when above optimal rate	Logical Error	● Low	Resolved
L-02	Migration Causes Unexecutable Orders	Unexpected Behavior	● Low	Acknowledged
L-03	Lacking Configuration Validations	Validation	● Low	Resolved
L-04	Stale Orders Allow For Short Term Risk Free Trades	Unexpected Behavior	● Low	Acknowledged
L-05	Timestamp Initialization Impacts Existing Orders	Unexpected Behavior	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-06	Lacking Deposit And Withdrawal Migrations	Configuration	● Low	Acknowledged
L-07	Atomic Withdrawals Cannot Be Simulated	Unexpected Behavior	● Low	Acknowledged
L-08	Longs Pay Higher Borrowing Fees As Price Increases	Unexpected Behavior	● Low	Resolved
L-09	Lacking timestampAdjustment Configurations	Configuration	● Low	Acknowledged
L-10	Redundant priceFeed Checks	Optimization	● Low	Resolved
L-11	Timestamp Adjustments DoS Atomic Withdrawals	Unexpected Behavior	● Low	Resolved
L-12	Shifts Are Allowed In The Same Market	Unexpected Behavior	● Low	Resolved
L-13	LPs May Avoid Losses With Atomic Withdrawals	Unexpected Behavior	● Low	Acknowledged
L-14	Outdated NatSpec	Documentation	● Low	Resolved
L-15	GM Oracle Salt Optimization	Optimization	● Low	Resolved
L-16	Unused Errors	Optimization	● Low	Resolved
L-17	Use Of Lagging validFromTimestamp	Unexpected Behavior	● Low	Resolved
L-18	Users Can use Shift to Bypass Disabled Features	Logical Error	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-19	setPositionImpactDistributionRate Missing Validation	Validation	● Low	Acknowledged
L-20	Inconsistent naming of function in key contract	Logical Error	● Low	Acknowledged
L-21	Migrating Orders Resets Cancellation Cooldown	Logical Error	● Low	Acknowledged
L-22	usageFactor Can Exceed 100%	Documentation	● Low	Acknowledged
L-23	Market Orders Can be Added to AutoCancel List	Unexpected Behavior	● Low	Resolved
L-24	Reference Price Check Bound Is Exceedingly Large	Configuration	● Low	Acknowledged

C-01 | External Call Gas Adjustment DoS's Liquidations

Category	Severity	Location	Status
Logical Error	● Critical	OrderUtils.sol: 384	Resolved

Description PoC

In the clearAutoCancelOrder function the cancelOrder function is invoked from within the OrderUtils library, however the cancelOrder function will account for an external delegatecall being made as it subtracts gasleft() / 63 from the startingGas amount. No external call will be made as the cancelOrder function is being called from within the context of the OrderUtils file and therefore the cancelOrder function will be inlined in the clearAutoCancelOrder function.

This errantly reduces the startingGas used to measure the gas expenditure for the keeper while cancelling autoCancel orders. As a result any attempt to cancel autoCancel orders will revert as the startingGas has been reduced such that it is now below the gasleft(). Therefore any positions with autoCancel orders cannot be closed as long as those orders exist, resulting in un-liquidatable positions.

Recommendation

Add a shouldAdjustStartingGas parameter to the cancelOrder function to account for when the cancelOrder function is invoked via delegatecall vs. function inlining.

Resolution

GMX Team: Resolved.

C-02 | AutoCancellation Prevents Liquidations

Category	Severity	Location	Status
Logical Error	● Critical	OrderUtils.sol:L215-L229	Resolved

Description

During liquidation `OrderUtils.executeOrder` will be called. This function will check if this is a decrease order and if `sizeInUsd` is 0 after the order execution, if so will call `clearAutoCancelOrders`. Indeed liquidation order is a decrease order and `sizeInUsd` after liquidation will be 0, hence AutoCancellation mechanism will always be triggered with liquidations.

AutoCancellation will check for all stop-loss/take-profit orders available for that position which can be as high as 10 in current configuration.

This cancellation can do possibly 10 callback calls with sending 2.000.000 gas for each. Additionally, for every cancellation the gas usage is around 600.000. When we also considered the fee refund mechanism's callback call which per call will send 500.000 gas, we can possibly reach total amount of 31.000.000 and more when considering liquidation's gas usage itself. So a liquidation order might require more than 31.000.000 gas which is more than block gas limit in avalanche and also can be possibly problematic in Arbitrum because it is expected from keepers to provide this amount of gas while it is not ensured the keeper provided sufficient amount of gas for all these actions.

Recommendation

Multiple steps are required to resolve the issue in its entirety:

1. Either decrease the max auto cancel amount or further restrict gas usage for cancellation callback. Combination of both can also be used. In the end, it is crucial to check maximum possible gas usage for AutoCancellation related actions.

2. Include auto cancellation's gas usage when checking keeper's provided gas amount especially when it is a liquidation order.

Resolution

GMX Team: Partially Resolved.

C-03 | shiftGasLimitKey Returns Incorrect Gas Limit Key

Category	Severity	Location	Status
Logical Error	● Critical	Keys.sol: 467	Resolved

Description

In the Keys.sol file, the shiftGasLimitKey function returns the WITHDRAWAL_GAS_LIMIT key instead of the SHIFT_GAS_LIMIT key.

Therefore the estimated execution fee for shift actions will be significantly smaller than it ought to be, as a shift includes not only a withdrawal but a deposit action as well.

This will lead to the protocol keeper being unexpectedly drained of the native token, potentially stopping execution on the exchange for a period of time. The gas draining can occur due to regular exchange usage or can be easily leveraged by an attacker to maliciously drain the keeper.

Recommendation

Return the SHIFT_GAS_LIMIT key in the shiftGasLimitKey function.

Resolution

GMX Team: Resolved.

H-01 | Borrowing Fees Increase Based On Incorrect Rate

Category	Severity	Location	Status
Logical Error	● High	ExecuteDepositUtils.sol: 103	Resolved

Description

When a position is updated the updateFundingAndBorrowingState function is called which updates CUMULATIVE_BORROWING_FACTOR based on the recent rate as well as the time since the last update. The rate of which the CUMULATIVE_BORROWING_FACTOR will increase is dependent on what percentage of the pools liquidity is being borrowed. The higher the percentage the higher the rate. By updating the CUMULATIVE_BORROWING_FACTOR before any changes to the state that could affect rate the borrowing fees can correctly be calculated.

The issue however is that this is not the case everywhere. When a user withdraws or deposits they will change the rate. As they withdraw the rate will increase and as they deposit the rate will decrease. However because the CUMULATIVE_BORROWING_FACTOR is not updated before a deposit/withdraw, the next time it is updated the rate will use the new value not the value that was actually representative of the elapsed time. Leading to excessive fees being charged if a withdraw occurs, or insufficient fees being charged if a deposit occurs.

With protocols integrating into GMX large deposits or withdraws will occur which will have a larger impact on the inaccuracy of the fees. The excessive fees being charged would lead to near liquidateable positions to become unexpectedly pushed to a liquidateable state. This step-wise jump in borrowing fees can also lead to arbitrage opportunities where attackers can profit off the inaccurate jump by making timely orders and deposits.

Recommendation

Update the Funding and Borrowing state early in the executeDeposit and executeWithdrawal functions.

Resolution

GMX Team: Resolved.

H-02 | Atomic Providers Cannot Be Configured

Category	Severity	Location	Status
Logical Error	● High	Timelock.sol: 163	Resolved

Description

In the `setAtomicOracleProviderAfterSignal` function the `isOracleProviderEnabledKey` is used instead of the `isAtomicOracleProviderKey`.

As a result the `atomicOracleProvider` value cannot be set, preventing any price feed provider from atomic withdrawal use, and disallowing the atomic withdrawal feature.

Recommendation

Change the `isOracleProviderEnabledKey` to the `isAtomicOracleProviderKey`.

Resolution

GMX Team: Resolved.

H-03 | Users Can Use Shift To Avoid Deposit Fees

Category	Severity	Location	Status
Logical Error	● High	ShiftHandler.sol:L66	Resolved

Description PoC

Users can avoid the SwapPricingType.TwoStep fee on deposits by utilizing shift and its lack of fees. Inside _executeDeposit, SwapPricingUtils.getSwapFees calculates the fee that users would pay for depositing into the exchange. Where there are 3 fees - TwoStep, Atomic, and 0 fee for shifts.

Users can abuse the lack of fee for shifts by simply front-running the keeper and sending their desired tokens to the shiftVault. When the keeper calls executeShift, these tokens would be accounted for as deposited from the shift when recordTransferIn is executed. This way, users can avoid paying the SwapPricingType.TwoStep fee.

For Example:

1. User makes a deposit of 1 USDC into the USDC:WETH vault.
2. User creates a shift for the same market.
3. User sees keeper TX and front-runs with 10,000 USDC and 2 WETH.
4. Keeper executes shift:
1. In the middle of the shift after the withdrawal, the tokens are recorded from the ShiftVault.
2. The recorded change is 10,001 USDC and 2 WETH.
5. The shift deposits the USDC and WETH while avoiding the fee.

Recommendation

Call shiftVault.recordTransferIn for the long and short tokens when starting executeShift to account for any tokens sent directly to it.

Resolution

GMX Team: Resolved.

H-04 | Cant set data stream through config

Category	Severity	Location	Status
Validation	● High	Config.sol: 119	Resolved

Description

When the setDataStream function is called there is a check that ensures the dataStream is not already set. However the check incorrectly does not reference the dataStore instead the check only uses the Keys.dataStreamIdKey(token) key. Which will never return 0. Which means the check will always fail and Data Streams will not be able to be added via the config.

Recommendation

Reference Data Store when checking if the stream has already been added.

Resolution

GMX Team: Resolved.

H-05 | GMOracleProvider Reverts Due To Incorrect Validation

Category	Severity	Location	Status
Logical Error	● High	GMOracleProvider.sol:L152-L184	Resolved

Description

GMOracleProvider get signed prices from Oracle keepers and tries to validate them with validateSigner(). What is expected from Order Keepers is providing both minPrices and maxPrices in ascending order for a token.

The problem occur because these minPrices and maxPrices are validated against their corresponding index in the signers and signatures array. But it is not guaranteed and in most cases won't be possible to both sort minPrices and maxPrices while protecting their corresponding signatures in the correct index. Consider the following scenario:

- Oracle Keeper 1 signs minPrice = 1003, maxPrice = 1010
- Oracle Keeper 2 signs minPrice = 1004, maxPrice = 1011
- Oracle Keeper 3 signs minPrice = 1005, maxPrice = 1009

Here when Order Keeper sort both prices in ascending order they will be sorted as follows:
minPrice = [Keeper1 minPrice, Keeper2 minPrice, Keeper3 minPrice]
maxPrice = [Keeper3 maxPrice, Keeper1 maxPrice, Keeper2 maxPrice]

Hence in validateSigner call, prices and respective signatures won't match and call will revert.
If on the other hand Order Keeper does not sort the order as above, then sorting check will fail and transaction will again revert.

Recommendation

Use old indexing system that matches min/maxPrice to their corresponding signers before validating signer.

Resolution

GMX Team: Resolved.

H-06 | Gas Validation Does Not Account For Callback Gas

Category	Severity	Location	Status
Logical Error	● High	GasUtils.sol: L63 and L81	Resolved

Description

When keeper performing the actions, if for any reason executions are failed, it is caught in their respective _handleError functions. These functions before checking the reason for execution error and handling the cancellation logic, calls gasUtils' validateExecutionErrorGas function to check if execution trace can be followed till the end with current gasLeft.

The problem is, the variable that is checked against gasLeft is MIN_HANDLE_EXECUTION_ERROR_GAS and it is configured as 1.200.000 and did not take into account cancellation's callback gas usage. So this check can pass while gas provided by keeper can be fully used in the concurrent process and that can lead to forwarding less than enough gas to callback contracts which would create unexpected silent reverts for systems integrating with GMX V2, which in many cases could cause a loss of funds or protocol disruption for those integrators.

The same situation also applies to getExecutionGas() and it's corresponding variable: MIN_HANDLE_EXECUTION_ERROR_GAS_TO_FORWARD which is configured as 1.000.000 Additionally REFUND_EXECUTON_FEE_GAS_LIMIT also is not accounted which should be accounted similarly.

Recommendation

When checking gas to see if it would be enough to handle executions, take into account the gas usage for callbacks.

Resolution

GMX Team: Resolved.

H-07 | MarketSwap Orders May Use Unexpected Prices

Category	Severity	Location	Status
Validation	● High	SwapOrderUtils.sol: 30	Partially Resolved

Description

In the processOrder function for swap orders, the price timestamp validation includes no maxOracleTimestamp validation for MarketSwap orders.

As a result a keeper may accidentally or maliciously execute a MarketSwap order when it is far past its request expiration age, with prices that are significantly unfavorable for the user.

Recommendation

When executing a MarketSwap order be sure to validate that the maxOracleTimestamp is not above the order's requestExpirationTime.

Resolution

GMX Team: Partially Resolved.

H-08 | Atomic Withdrawal Feature Unsable

Category	Severity	Location	Status
Logical Error	● High	Oracle.sol:L224	Resolved

Description

During executeAtomicWithdrawal modifier withOraclePrices is used which when we follow the call trace we will reach the _validatePrices function in Oracle.sol. In this function we check the provider for the given tokens to validate that it is indeed the same provider given by keeper.

The problem is, the normal providers for tokens will be the dataStreamProvider while for atomic withdrawals they will be the priceFeedProvider. Hence keeper provided provider won't match this provider in dataStore which will lead to reverts for all atomic withdrawal calls.

Recommendation

If the action is atomic withdrawal instead of comparing the provided provider with oracleProviderForTokenKey, compare it with isAtomicOracleProviderKey.

Resolution

GMX Team: Resolved.

H-09 | Migration Inconsistencies

Category	Severity	Location	Status
Logical Error	● High	Global	Acknowledged

Description

According to documentation there is a time where both old contracts and new contracts are live and used by keepers. This can create massive inconsistency in the system such as:

1. Borrowing fee calculations are different between systems which will lead to both different borrowing fee's for users between systems and also different market token pricing's for the system which can have catastrophic effect.

2. While old system continue to send excess fees to `account`, new system will send them to `receiver`. Which can be especially problematic for integrators who changed their system according to new implementation and can't send excess fees to `receiver` anymore.

3. Take profit and stop-loss orders that are opened with new contracts will be added to autoCancelList but if the position is closed/liquidated with the old keeper, this lost won't be cleared. Which in turn, users can experience unexpected position openings in the future if they continue to use the system.

Recommendation

Try to not use both contracts at the same time, if they will be used, be sure to not set `optimalBorrowingFactor` until old system abandoned and also inform users and integrations about inconsistencies that may arise because of this situation.

Resolution

GMX Team: Acknowledged.

M-01 | makeExternalCalls Unexpected Funds Receiver

Category	Severity	Location	Status
Logical Error	● Medium	ExternalHandler.sol: 51	Acknowledged

Description

When makeExternalCalls is used a batch of target contracts are called. This is used to allow users to interact with external contracts to perform a variety of operations.

After the contracts are called makeExternalCalls will loop through an array of refund tokens and send each token to a desired recipient.

The issue is that in cases where two recipients are expected to receive the same token the first recipient will receive 100% of the tokens while the second recipient will receive nothing.
This is because the amount sent to each recipient is based on the balanceOf for the specific token, which ensures the entire balance will be used on the first recipient. Resulting in some address not receiving their expected funds.

Recommendation

Iterate through refundToken and check that there are no duplicate addresses in the array.

Resolution

GMX Team: Acknowledged.

M-02 | Attack can game price impact at users expense

Category	Severity	Location	Status
Logical Error	● Medium	GasUtils.sol: 135	Acknowledged

Description

Price impact is used to incentivize bringing markets to a balanced state. However, this can be gamed by bad actors who want to profit off other users.

An attacker can monitor transactions and when they see a deposit that is going to bring the market to a balanced state they can send multiple orders that do the same thing as the victim. The attacker can set the min output amount to a value greater than the input amount so that the only way the order will execute is if the order obtained the positive price impact, the rest would revert costing the user nothing more than a portion of the execution fee.

After the attackers order is executed the victims order will also execute, but they will experience negative price impact since their order is moving the price away from balanced. The attacker can then withdraw, bringing the pool back to balance again.

The reason this attack can be effective despite it not being certain that the attackers order will execute first is because the attacker can create many orders and only one needs to beat the victim. By sending more than one order the attacker is increasing the chance that it will be executed first and making this attack both likely to succeed and profitable in certain situations.
By the end of the attack the attacker was able to obtain positive price impact twice where one of those times was at the expense of the other user.

Recommendation

Consider refunding less of the execution fee upon cancellation to make these type of attacks unprofitable.

Resolution

GMX Team: Acknowledged.

M-03 | MAX_AUTO_CANCEL_ORDERS Update Risk

Category	Severity	Location	Status
Unexpected Behavior	● Medium	AutoCancelUtils.sol: L26-L31	Acknowledged

Description

When a position is completely closed (via user or by liquidation), user's orders in autoCancelList will be cancelled via providing MAX_AUTO_CANCEL_ORDERS as a max index to auto cancel list. If that variable changes to a variable that is less, then some orders will stay at the unreachable part of the autoCancel list.

If user continues to use the system, these orders can be executed when they are not expecting. Since auto cancellation process is gas intensive, MAX_AUTO_CANCEL_ORDERS is a variable that can be changed more than other variables to limit the gas usage of a single call. Hence it is very possible for this problem to occur in production.

Recommendation

In the case of the aforementioned variable is changed, inform users so that they can cancel their orders.

Resolution

GMX Team: Acknowledged.

M-04 | uiFee is Taken Twice During Shift

Category	Severity	Location	Status
Logical Error	● Medium	ShiftUtils.sol	Resolved

Description

During shift, uiFee taken twice both in withdrawal and also in deposit. Since both deposit and withdrawal done in a single action, it should be taken only once.

Recommendation

Include the uiFee on either the deposit or the withdraw, but not both.

Resolution

GMX Team: Resolved.

M-05 | Dangerous Atomic Withdrawal Invocation Pattern

Category	Severity	Location	Status
Unexpected Behavior	● Medium	WithdrawalHandler.sol	Resolved

Description

The executeAtomicWithdrawal function is intended to be called by users or integrators on the WithdrawalHandler after depositing market tokens into the WithdrawalVault through the ExchangeRouter. However users are not able to use a multicall on the ExchangeRouter to do so seamlessly in a single transaction, as the executeAtomicWithdrawal function is only available through the WithdrawalHandler contract.
This promotes a dangerous pattern where users may send their market tokens to the WithdrawalVault through the ExchangeRouter and call the executeAtomicWithdrawal function on the WithdrawalHandler in separate transactions. In this case the user is exposed to risk of total loss of funds if another actor frontruns their second transaction to executeAtomicWithdrawal, by malicious intent or on accident.

Recommendation

Consider only exposing the executeAtomicWithdrawal functionality through the ExchangeRouter, this way it can be invoked through a multicall. Additionally, be sure to document this risk to users and integrators, advising them to use the multicall feature.

Resolution

GMX Team: Resolved.

M-06 | Shifts Within A Virtual Inventory Unfairly Punished

Category	Severity	Location	Status
Unexpected Behavior	● Medium	Global	Resolved

Description

When shifting between two markets in the same virtual inventory the shift will often experience negative impact from the virtual inventory even though the shift did not cause an imbalance in the virtual inventory token amounts.

Consider the following scenario:

- Market A has a longTokenUsd of 200 and a shortTokenUsd of 300
- Market B has a longTokenUsd of 505 and a shortTokenUsd of 500
- A virtual inventory is comprised of Market A and Market B, with an aggregate longTokenUsd of 705 and shortTokenUsd of 800
- Bob shifts 20% of the Market A marketToken supply to Market B, 40 longTokenUsd and 60 shortTokenUsd are shifted
- During the withdrawal the virtual inventory diff goes from 705 - 800 = -95 to 665 - 740 = -75
- During the deposit the MarketB diff goes from 505 - 500 = 5 to 545 - 560 = -15, while the virtual inventory diff goes from 665 - 740 = -75 to 705 - 800 = -95
- The net virtual inventory diff stays the same from the start of the shift to the end of the shift, however the user receives increased negative impact because the deposit creates a larger imbalance in the virtual inventory than in Market B.

In this scenario the user is not causing any imbalance to the virtual inventory and should therefore not be negatively impacted by the virtual inventory diff during deposit.

Recommendation

Consider ignoring price impact from the virtual inventory when shifting between two markets that are in the same virtual inventory, as this action will never cause a further imbalance in the virtual inventory. A more complete alternative would be to consider the net virtual inventory diff created by an entire shift action, this way the virtual inventory diff created by uiFees and other potential balance changes can be accounted for.

Resolution

GMX Team: Resolved.

M-07 | setFundingRate Unexpectedly Changes Funding

Category	Severity	Location	Status
Unexpected Behavior	● Medium	Config.sol: 59	Acknowledged

Description

The setFundingRate function allows a permissioned address to update the maxFundingRate, which potentially caps the existing funding rate for pending funding fees.

This action would affect funding fees that have accumulated in the past causing unexpected funding changes for users, which could ultimately lead to accounts being subject to unexpected liquidation as a result of receiving less funding fees than expected.

Recommendation

Update the funding state before updating the maxFundingRate similar to the setPositionImpactDistributionRate function distributes the impact pool.

Resolution

GMX Team: Acknowledged.

M-08 | MEV Bribes Allow Griefing Of Keepers

Category	Severity	Location	Status
Protocol Manipulation	● Medium	Global	Acknowledged

Description

On Avalanche C-Chain a user can use providers such as flashbots or snowsight to set the gas price to the lowest acceptable gas price and still have their transaction executed via a bribe.

Inside of `validateExecutionFee()`, the validation for the execution fee checks if execution fee is less than `gasLimit` * `tx.gasprice`. Setting a lower `tx.gaprice` will allow a user to pay less than the expected amount for an execution fee and force the keeper to draw on treasury reserves to subsidize the transaction.

It may be possible for a malicious actor to submit many orders this way in order to grief the keepers. Or users may use this to pay less execution fees on the exchange consistently.

Recommendation

Set a value for the lowest acceptable gas price, and verify that the value of `tx.gasprice` is greater than this value in `validateExecutionFee()`.

Resolution

GMX Team: Acknowledged.

M-09 | Refund gas limit is not accounted for

Category	Severity	Location	Status
Validation	● Medium	GasUtils.sol:L165	Acknowledged

Description PoC

The payExecutionFee function refunds the leftover gas to the user **before** executing the external callback to refundExecutionFee. This gas refund doesn't take into account the fact that refundExecutionFee can use up to 500k gas. This will enable users to grief keepers, making their TX unprofitable.

For Example:

1. User makes an order with 3m gas as execution fee.
2. Keeper executes that order with 3m gas (equal execution fee).
3. We reach payExecutionFee, where up to now 2m gas is used.
4. Keeper is payed 2m and the user is refunded 1m.
5. The refundExecutionFee triggers wasting 500k gas.

In the current scenario the user only paid 2m gas for his order, but costed the keeper 2.5m gas.

Recommendation

Increase EXECUTION_GAS_FEE_BASE_AMOUNT in order for adjustGasUsage to calculate the keeper gas properly.

Resolution

GMX Team: Acknowledged.

L-01 | Kink Borrowing Yields Unexpected Rate

Category	Severity	Location	Status
Logical Error	● Low	MarketUtils.sol: 2425	Resolved

Description

The kink borrowing factor is meant to increase the amount a user pays in fees when the usage exceeds the optimal threshold. Specifically the borrow rate changes from a base factor, to an above optimal factor.

The issue here is that the base is applied to the entire `usageFactor` not just the portion that is optimal. What this means is that when the above optimal factor is applied to the portion that is beyond the threshold, that portion is double charged. Once at the base factor and then again at the above optimal factor.

The current kink charge may be an unexpected jump in borrowing fees for users.

Recommendation

Consider only applying the base factor to the portion that is at or below the threshold. Then apply the above optimal threshold to the portion is above the threshold.

Resolution

GMX Team: Resolved.

L-02 | Migration Causes Unexecutable Orders

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description

TimestampInitializer.sol will set all orders that are in ORDER_LIST and all positions that are in POSITION_LIST's related timestamps (updatedAt, increasedAt, decreasedAt) to current timestamp if they have 0 as a value as a final migration step. This will make all order's that are created at the past and not executable(because REQUEST_EXPIRATION_TIME has passed since their latest update), and doesn't cancelled yet, executable.

Hence users can experience unexpected order executions because against the expected workflow, orders with REQUEST_EXPIRATION_TIME passed will be executed.

Recommendation

Consider specifically avoiding these orders when updating the timestamps, if they will be updated, inform users beforehand.

Resolution

GMX Team: Acknowledged.

L-03 | Lacking Configuration Validations

Category	Severity	Location	Status
Validation	● Low	Config.sol: 499-501	Resolved

Description

Newly allowed configuration variables including `OPTIMAL_USAGE_FACTOR`, `BASE_BORROWING_FACTOR`, and `ABOVE_OPTIMAL_USAGE_BORROWING_FACTOR` have not been added to function `_validateRange()`.

This directly contrasts with similar configuration variables such as `BORROWING_FACTOR` which is validated in `_validateRange` to not be more than 100%. Consequently, invalid values may be set for such factors.

Recommendation

Add the new factors to `_validateRange` to ensure they do not exceed 100%, similar to `BORROWING_FACTOR`

Resolution

GMX Team: Resolved.

L-04 | Stale Orders Allow For Short Term Risk Free Trades

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description PoC

With the new oracle pricing mechanism, market orders can only be executed with prices in the range [orderUpdatedAt, orderUpdatedAt + requestExpiration]. As a result, if a market order is not executed in a timely manner it can only be executed with outdated prices, which allows malicious actors to make risk free traders. Consider the following scenario based on current configurations:

- maxPriceAge is 5 minutes
- requestExpiration is 5 minutes
- orderUpdatedAt is at t = 100
- The request expiration is at t = 105
- The current time is t = 108

In this scenario the keeper may only execute the order with prices from the range [100, 105], meanwhile the current price is at t = 108. Additionally, since the market order has passed the request cancellation period, the user may cancel their order if price has not moved in a direction that benefits them.

The most straightforward application of this is a swapOrder with a swapPath which takes advantage of these outdated prices. The baseMarketConfig has a swapFeeFactorForPositiveImpact of 0.05% and a swapFeeFactorForNegativeImpact of 0.07%, assuming the swap receives the worse feeFactor, the swap would have to net > ~0.10% gain as a result of the outdated prices to be reasonably profitable.
An analysis of minute candles for ETH/USD shows that ETH often moves by 0.10% or more in a single minute, given the price can be stale by up to 5 minutes it is possible that swaps could arbitrage a 0.20%+ gain for the user.

Recommendation

Consider restricting the execution of market orders, deposits, and withdrawals past their request expiration time. Otherwise consider introducing a stale order threshold time where an order can still be executed after the request expiration time, but not after the stale order time such that the prices do not have a chance to grow stale by a number of minutes.
Currently the chainlink reference oracle will not prevent such an arbitrage as the maxRefPriceDeviationFactor is 50%, another solution could be to make this deviation factor much smaller — though this may introduce unnecessarily tight validation on prices for other pricing mechanisms.

Resolution

GMX Team: Acknowledged.

L-05 | Timestamp Initialization Impacts Existing Orders

Category	Severity	Location	Status
Unexpected Behavior	● Low	TimestampInitializer.sol	Acknowledged

Description

The TimestampInitializer contract attempts to set the timestamp on all open orders and positions to the current Arbitrum timestamp. However, this may cause orders that would have executed prior to the timestamp transition to now fail in function validateOracleTimestamp() or be executed at worse prices.

Consider the following scenario:
1. Bob increases their long position at block = 1; time = 1 at price $5000
2. Bob sees price is going downwards, so Bob submits a SL order at block = 5; time = 5 for price $4,500
3. Timestamps are initialized at time = 10 when price is $4,400. Both the orderUpdatedAtTime and positionDecreasedAtTime are now time=10.
4. The SL order cannot be executed at price $4,500 but at price $4,400, since the range when the $4,500 price was available for execution was effectively erased with the timestamp initialization

Recommendation

Rather than assigning the current block timestamp to outstanding orders and positions, consider converting the existing block numbers for orderUpdatedAtBlock, positionIncreasedAtBlock, and positionDecreasedAtBlock to their respective block.timestamps by passing in a list of the correct timestamps for each order.

Resolution

GMX Team: Acknowledged.

L-06 | Lacking Deposit And Withdrawal Migrations

Category	Severity	Location	Status
Configuration	● Low	Global	Acknowledged

Description

While there exist migrations for orders and positions there are none for deposits and withdrawals. While deposits and withdrawals will not be waiting in their respective stores for long, it may be possible that some are left in flight while the contract upgrades are made.

These old deposits and withdrawals which do not have an `updatedAtTime` cannot be executed as they will fail the price validations. They will not be cancelled by the keeper as the revert will be from an oracle error, therefore the deposits and withdrawals will continue to exist in their respective stores until they are cancelled manually.

Some integrating systems, such as Umami's GMI index, do not currently implement logic to call the `cancelDeposit` or `cancelWithdrawal` functions on the `exchangeRouter`. Thus if any deposits or withdrawals would be caught in this state they would at worst cause a loss of funds as the orders would not be cancellable and at best require a logic upgrade to be able to cancel these orders manually.

Recommendation

Be sure to only conduct the contract upgrade while there are no pending deposits and withdrawals. Otherwise consider implementing migrations for deposits or withdrawals.

Resolution

GMX Team: Acknowledged.

L-07 | Atomic Withdrawals Cannot Be Simulated

Category	Severity	Location	Status
Unexpected Behavior	● Low	WithdrawalHandler.sol: 150	Acknowledged

Description

In the WithdrawalHandler, the simulateExecuteWithdrawal function has been updated to accept a swapPricingType parameter, presumably to be able to simulate normal two-step withdrawals as well as atomic withdrawals.

However the simulateExecuteWithdrawal function cannot be used to simulate atomic withdrawals as these withdrawals have not been created in the WithdrawalStore and the function attempts to retrieve the withdrawal via a withdrawalKey.

Recommendation

Consider making a separate simulation function to allow users and integrators to simulate atomic withdrawals.

Resolution

GMX Team: Acknowledged.

L-08 | Longs Pay Higher Borrowing Fees As Price Increases

Category	Severity	Location	Status
Unexpected Behavior	● Low	MarketUtils.sol: 503	Resolved

Description

In markets where longs are backed by the same token as the index, or when the backing longToken is correlated with the index, as the index price and the reservedUsd increases so does the maxReservedUsd.

In fact the maxReservedUsd will increase more than the reservedUsd does, as long as the reservedUsd is less than the maxReservedUsd.

Therefore traders with longs will pay a lower borrowing rate as the price of the index increases. However with the new kink borrowing model, since the openInterestLimit is capped at a fixed USD maxOpenInterest value, when the maxOpenInterest is the constricting limit traders may pay a significantly higher borrowing rate as the index price increases.

Recommendation

Consider if this is the expected behavior, if so document it so trader's can be aware of these borrowing fee dynamics.

Resolution

GMX Team: Resolved.

L-09 | Lacking timestampAdjustment Configurations

Category	Severity	Location	Status
Configuration	● Low	Global	Acknowledged

Description

dataStreams will be configured with a timestampAdjustment to account for price latency from differing sources, however in the setDataStream, signalSetDataStream, and setDataStreamAfterSignal functions there is no logic to allow the configuration of a timestampAdjustment for a dataStream.

While the ORACLE_TIMESTAMP_ADJUSTMENT key is an allowed base key, these functions may elect to offer accessibly configuration of the adjustment as necessary.

Recommendation

Add configuration support for the timestampAdjustment in the setDataStream function.

Resolution

GMX Team: Acknowledged.

L-10 | Redundant priceFeed Checks

Category	Severity	Location	Status
Optimization	● Low	Oracle.sol: 244	Resolved

Description

In the _validatePrices function the reference chainlink price feeds are validated for the maxRefPriceDeviationFactor even when the price provider is the ChainlinkPriceFeedProvider.

Recommendation

Do not perform the maxRefPriceDeviationFactor check when the price provider is the ChainlinkPriceFeedProvider as this check is redundant.

Resolution

GMX Team: Resolved.

L-11 | Timestamp Adjustments DoS Atomic Withdrawals

Category	Severity	Location	Status
Unexpected Behavior	● Low	Oracle.sol: 238	Resolved

Description

The timestampAdjustment is not intended to be used with the ChainlinkPriceFeedProvider. However if it is accidentally configured for a ChainlinkPriceFeedProvider then this provider cannot be used with atomic withdrawals as the price must have a timestamp of the current block.timestamp.

Recommendation

Add validation such that the timestampAdjustment cannot be configured to be nonzero when the provider is a ChainlinkPriceFeedProvider.

Resolution

GMX Team: Resolved.

L-12 | Shifts Are Allowed In The Same Market

Category	Severity	Location	Status
Unexpected Behavior	● Low	ShiftUtils.sol: 67	Resolved

Description

When creating a shift there is no validation that prevents a user from shifting out of and back into the same market. While no high impact outcome has been identified, this action serves no purpose and increases the likelihood of a potentially unexpected state.

Recommendation

Out of an abundance of caution, consider validating that the from market is not the same as the to market when creating a shift.

Resolution

GMX Team: Resolved.

L-13 | LPs May Avoid Losses With Atomic Withdrawals

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description

In the event that an insolvent liquidation occurs, there is a stepwise decrease in the value of a `MarketToken`, as the accounting system realizes that the liquidated position cannot cover the losses it had tracked in the `getPoolValueInfo` function.

Informed depositors may observe this liquidation and frontrun it to avoid such losses with an `atomicWithdrawal`. This way more of the value losses impact the other depositors who have not withdrawn, and the user avoids any losses due to the insolvency.

Recommendation

Be aware of this manipulation and ensure the atomic withdrawal fee is maintained at a high rate to disincentivize this gaming.

Resolution

GMX Team: Acknowledged.

L-14 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	● Low	GmOracleUtils.sol: 24	Resolved

Description

The NatSpec for the validateSigner function is outdated as it still references several parameters such as the blockHash, minOracleBlockNumber, and maxOracleBlockNumber which are no longer used.

Recommendation

Update the NatSpec for the validateSigner function to match the current parameters.

Resolution

GMX Team: Resolved.

L-15 | GM Oracle Salt Optimization

Category	Severity	Location	Status
Optimization	● Low	GmOracleProvider.sol: 175	Resolved

Description

In the getOraclePrice function the _getSalt function is called in a loop for every signer, however the value will not change within the same transaction, therefore the result of _getSalt can be cached outside of the signer loop.

Recommendation

Cache the result of the _getSalt function before the signer loop to save gas from needlessly re-computing it.

Resolution

GMX Team: Resolved.

L-16 | Unused Errors

Category	Severity	Location	Status
Optimization	● Low	Errors.sol	Resolved

Description

In the `Errors.sol` file there are several errors which are no longer used in the GMX V2 codebase:

- `CouldNotSendNativeToken`
- `EmptyCompactedPrice`
- `EmptyCompactedBlockNumber`
- `EmptyCompactedTimestamp`
- `UnsupportedOracleBlockNumberType`

Recommendation

Remove these unused errors from the `Errors.sol` file.

Resolution

GMX Team: Resolved.

L-17 | Use Of Lagging validFromTimestamp

Category	Severity	Location	Status
Unexpected Behavior	● Low	ChainlinkDataStreamProvider.sol: 84	Resolved

Description

The ChainlinkDataStreamProvider uses the report.validFromTimestamp as the validated price timestamp. However this price is from the earliest time in the range, this is in contrast to the GmOracleProvider which correctly uses the timestamp where the price was aggregated from. For example consider the following prices reported by chainlink's data stream:

1. t = 150, validFromTimestamp = 148, observationsTimestamp = 150
2. t = 155, validFromTimestamp = 151, observationsTimestamp = 155
3. t = 158, validFromTimestamp = 156, observationsTimestamp = 158

An order submitted at t = 153 should be allowed to use the price which is observed at t = 155, however it will not be usable as the validFromTimestamp of 151 is used.

Recommendation

Use the report.observationsTimestamp for the validated price timestamp instead of the report.validFromTimestamp.

Resolution

GMX Team: Resolved.

L-18 | Users Can use Shift to Bypass Disabled Features

Category	Severity	Location	Status
Logical Error	● Low	ShiftUtils.sol: 142	Acknowledged

Description

The validation for if a market is disabled to be withdrawn from or deposited into occurs in WithdrawalHandler::_executeWithdrawal() and DepositHandler::_executeDeposit().

When using shift it calls ExecuteWithdrawalUtils::executeWithdrawal() and ExecuteDepositUtils::executeDeposit(), which is past when those checks occur in the withdraw and deposit flow. This allows a user to withdraw and deposit from markets that do not allow withdrawals or deposits.

Recommendation

Before calling ExecuteWithdrawalUtils::executeWithdrawal() and ExecuteDepositUtils::executeDeposit() inside of ShiftUtils::executeShift(), check if the withdraw and deposit feature is disabled for that market. Alternatively, verify that the shift feature is disabled if withdraw or deposit feature is disabled.

Resolution

GMX Team: Acknowledged.

L-19 | setPositionImpactDistributionRate Missing Validation

Category	Severity	Location	Status
Validation	● Low	Config.sol: 200	Acknowledged

Description

Unlike the other functions in the config contract. the setPositionImpactDistributionRate function does not validate the provided parameters. Such as positionImpactPoolDistributionRate.

Recommendation

To ensure that there are no unexpected values set when calling the setPositionImpactDistributionRate consider adding checks for the positionImpactPoolDistributionRate.

Resolution

GMX Team: Acknowledged.

L-20 | Inconsistent naming of function in key contract

Category	Severity	Location	Status
Logical Error	● Low	Keys.sol: 756-767	Acknowledged

Description

In the Keys contract the function are used to retrieve specific keys. Each function follows a specific naming convention of using the name of key and then the word key Example: function oracleProviderForTokenKey(address token).

However there are two functions that do not follow this pattern tokenTransferGasLimit and savedCallbackContract.

Recommendation

Consider changing the naming of these two functions to be consistent with the the rest of the functions in the contract

Resolution

GMX Team: Acknowledged.

L-21 | Migrating Orders Resets Cancellation Cooldown

Category	Severity	Location	Status
Logical Error	● Low	TimestampInitializer.sol: 47	Acknowledged

Description

When a order gets migrated the timestamp will be updated. Because of this users who were previously able to cancel their orders will have to wait for an additional period of time to cancel.

Recommendation

Document that the cancelling order cool down will be reset upon migration.

Resolution

GMX Team: Acknowledged.

L-22 | usageFactor Can Exceed 100%

Category	Severity	Location	Status
Documentation	● Low	MarketUtils.sol: 2407	Acknowledged

Description

The `usageFactor` in `getKinkBorrowingFactor` can exceed 100% as the prices of collateral and index tokens fluctuates.

Although `getOpenInterestLimit` includes a precaution to ensure `maxReservedUsd` is at or below 100%, there is still a possibility that a sudden price movement, when the market is near full utilization, could cause `reservedUsd` and `openInterestLimit` to push the `usageFactor` above 100%. Example:

1. `usageFactor - 80%`
2. The pool is currently at 79%, but a drop in collateral price pushes utilization to 81%
3. `getKinkBorrowingFactor` will then calculate `usageFactor` to be `> 1e18`

Recommendation

Consider documenting this behavior so that users are aware that the usage factor can go above 100% and result in higher than expected fees.

Resolution

GMX Team: Acknowledged.

L-23 | Market Orders Can be Added to AutoCancel List

Category	Severity	Location	Status
Unexpected Behavior	● Low	OrderUtils.sol:L395:L397	Resolved

Description

Documentation related to update specifies that take profits and stop-loss orders with autoCancel flag will be added to autoCancelList.

But it only checks if it is a decrease order before adding to the list. Which means MarketDecrease orders can be added to the list. This allows market orders to be cancelled within the requestExpiration window.

Recommendation

In updateAutoCancelList() aside from checking if the order is decrease order, also check if it is a market order. If it is a market order, don't add to the autoCancelList.

Resolution

GMX Team: Resolved.

L-24 | Reference Price Check Bound Is Exceedingly Large

Category	Severity	Location	Status
Configuration	● Low	Oracle.sol:L246	Acknowledged

Description

All prices will be validated against the Chainlink priceFeedPrice if they have address configured for priceFeeds. Currently this check allows price differences between oracles up to 50%.

Chainlink priceFeeds have 0.5%-2% deviation threshold which if a price of the commodity changes at least as much as deviation threshold, price will be updated. So a lot lower threshold for MAX_ORACLE_REF_PRICE_DEVIATION_FACTOR can be provided to be sure prices are in accepted ranges for different oracle usages.

Recommendation

Decrease MAX_ORACLE_REF_PRICE_DEVIATION_FACTOR to a reasonable value such that it won't revert unnecesarily but it can catch malicious/stale prices.

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 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