GA GUARDIAN

GMX

GLV GM Token Vault

Security Assessment

September 3rd, 2024

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Osman Ozdemir, Kiki, Wafflemakr, Mark Jonathas

Client Firm GMX

Final Report Date September 3, 2024

Audit Summary

GMX engaged Guardian to review the security of its liquidity vault for GM tokens, allowing seamless shifting of liquidity across markets according to their utilization. From the 30th of July to the 12th of August, a team of 6 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/glv-pocs

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics
Commit(s)	Initial : 35bd45ab514e6432497817205a3415e736df8d4e Final: 87a91148348952c5ff5b8fcb1bf6078342141f47

Audit Summary

Delivery Date	September 3, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
High	2	0	0	0	0	2
Medium	12	0	0	3	0	9
Low	31	0	0	11	0	20

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
- Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	GLV Arbitraged With pnlToPoolFactor	Logical Error	Critical	Resolved
H-01	Execution fee locked in router on cancellation	Logical Error	High	Resolved
H-02	GLV Callbacks Do Not Validate Remaining Gas	Validation	High	Resolved
M-01	Excess execution fee will be required	Logical Error	Medium	Resolved
M-02	GLV Withdrawal Callback Gas Cost Ignored	Logical Error	Medium	Resolved
M-03	`executionFee` Should Be Updated	Logical Error	Medium	Resolved
M-04	GLV Used To Exit Illiquid Markets	Logical Error	Medium	Acknowledged
M-05	GLV Actions Cannot Be Simulated	Logical Error	Medium	Resolved
M-06	GLV Used For Atomic Withdrawals	Gaming	Medium	Resolved
M-07	GLV Trapped Funds Upon Disabled	DoS	Medium	Resolved
M-08	GLV Read-only Reentrancy Risk	Reentrancy	Medium	Resolved
M-09	User GLV Deposits Errantly Maximized	Logical Error	Medium	Resolved
M-10	Virtual Inventory Ignored On Withdrawal	Logical Error	Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-11	GLV Oracle Count Wrong For Homogenous Markets	Unexpected Behavior	Medium	Acknowledged
M-12	Insufficient Gas Forwarded For Refund	Validation	Medium	Resolved
L-01	Execution Fees Increased By Empty Markets	Logical Error	Low	Resolved
L-02	Feature validation occurs before try-catch	Logical Error	Low	Resolved
L-03	DoS Via Type Casting	DoS	Low	Acknowledged
L-04	Unused `marketCount` Variable	Superfluous Code	Low	Resolved
L-05	GLV Deposit Config Added Twice	Superfluous Code	Low	Resolved
L-06	GM Deposits With False isMarketTokenDeposit	Validation	Low	Acknowledged
L-07	No Keeper Incentives To Create Shifts	Logical Error	Low	Acknowledged
L-08	Unused Struct Params	Superfluous Code	Low	Acknowledged
L-09	Shifting Is Allowed To Same Market	Validation	Low	Acknowledged
L-10	Cancelling Deposits Unnecessarily Transfers Tokens	Optimization	Low	Resolved
L-11	Malicious Code Can Be Set In Token Name/Symbol	XSS	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-12	Attacker can extract value from shifts	Warning	Low	Acknowledged
L-13	Missing function for feature key	Logical Error	Low	Resolved
L-14	Unused errors	Superfluous Code	Low	Resolved
L-15	GLV Deposit Creation DoS	DoS	Low	Acknowledged
L-16	GLV GM Balance Validation Minimizes Value	Validation	Low	Resolved
L-17	Users Charged For Oracles On Cancel	Logical Error	Low	Acknowledged
L-18	Funds Are Not Synced In The GLV Vault	Warning	Low	Resolved
L-19	GLV Deposits Incorrectly Value Fees	Logical Error	Low	Resolved
L-20	Incorrect Param Used in Event	Logical Error	Low	Resolved
L-21	Superfluous receivedUsd Value	Optimization	Low	Resolved
L-22	Superfluous Imports	Optimization	Low	Resolved
L-23	Lacking GlobalNonReentrant Modifier	Reentrancy	Low	Resolved
L-25	Errant GLV Keys	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-26	GLV Salt Lookup Not Cleared	Logical Error	Low	Acknowledged
L-27	Misleading setMarketTokenAmount Function	Typo	Low	Resolved
L-28	Lacking GM Market Count Cap	Validation	Low	Resolved
L-29	block.timestamp Usage	Suggestion	Low	Resolved
L-30	Users can imbalance glv to reduce glv yield	Logical Error	Low	Acknowledged
L-31	Withdrawal/Shift count reserved GM as available	Logical Error	Low	Acknowledged
L-32	Deposits With Zero Short Token Fail	Logical Error	Low	Resolved

C-01 | GLV Arbitraged With pnlToPoolFactor

Category	Severity	Location	Status
Logical Error	● Critical	Global	Resolved

Description PoC

Upon deposits to GLV the GM tokens in the vault are valued using the MAX_PNL_FACTOR_FOR_DEPOSITS, and on withdrawals the GM tokens in the vault are valued using the MAX_PNL_FACTOR_FOR_WITHDRAWALS.

The MAX_PNL_FACTOR_FOR_DEPOSITS will value GM tokens at a lower value than the MAX_PNL_FACTOR_FOR_WITHDRAWALS as the trader's PnL is capped using a higher percentage, resulting in more trader profits being allowed using the deposit pnlToPoolFactor measurement.

In the existing GMX V2 system this is not exploitable since withdrawals are not allowed if the market is over the MAX_PNL_FACTOR_FOR_WITHDRAWALS ratio. However in GLV there is no constraint on withdrawing a different GM market using your GLV tokens and benefiting from this arbitrage in total GLV value between deposits and withdrawals.

For example:
• GLV has GM A and GM B
• The GM A MAX_PNL_FACTOR_FOR_WITHDRAWALS is 40% and MAX_PNL_FACTOR_FOR_DEPOSITS is 60%
• GM A has +$500,000 of pending trader PnL and the pnlToPoolFactor is currently 50%
• User A deposits GM B to the GLV, the GLV is valued using MAX_PNL_FACTOR_FOR_DEPOSITS which allows for the full trader pnl of $500,000
• User A then withdraws GM B from the GLV, the GLV is valued using MAX_PNL_FACTOR_FOR_WITHDRAWALS which allows for only $400,000 of trader pnl, which values the GM A tokens higher.

As a result User A receives more GM B out of the withdrawal than they had initially deposited because of the arbitrage between the deposit factor and withdrawal factor.

Recommendation

Consider valuing all GM tokens in the GLV using the MAX_PNL_FACTOR_FOR_DEPOSITS even on withdrawals. This way the value of the GM tokens is minimized when the pnlToPoolFactor is currently above the withdrawal factor. This would inaccurately account for the value out of the GM tokens a user would receive if they withdrew those tokens, however those tokens would not be withdrawable if they are above this factor anyways.

Alternatively, consider disallowing all GLV withdrawals when any one of the market tokens is above it's MAX_PNL_FACTOR_FOR_WITHDRAWALS.

Resolution

GMX Team: Resolved.

H-01 | Execution fee locked in router on cancellation

Category	Severity	Location	Status
Logical Error	High	GlvHandler.sol: 123-224	Resolved

Description

When a user cancels a GLV deposit/withdraw the keeper is set to `msg.sender` instead of account. The `msg.sender` in this case would be the `GlvRouter` contract. Resulting in the keeper portion of the execution fee being sent to the router instead of the user who initiated the cancellations.

Recommendation

Pass in `account()` instead of `msg.sender` when a user initiates a cancellation.

Resolution

GMX Team: Resolved.

H-02 | GLV Callbacks Do Not Validate Remaining Gas

Category	Severity	Location	Status
Validation	High	CallbackUtils.sol	Resolved

Description

In the `afterGlvDepositExecution`, `afterGlvDepositCancellation`, `afterGlvWithdrawalExecution`, and `afterGlvWithdrawalCancellation` functions the `validateGasLeftForCallback` is not performed as it is in all other callbacks.

As a result these callbacks may forward less than the expected callback gas limit and cause integrating systems to silently revert, leading to loss of funds in these systems. Specifically, in validation done in the respective `_handleError` functions is against the `MIN_HANDLE_EXECUTION_ERROR_GAS` which is configured as 1,200,000 and does not take into account cancellation's callback gas usage.

Therefore this check can pass while gas provided by keeper is insufficient for the cancellation + callback gas which will lead to forwarding less than enough gas to the callback contract.
Without this validation keepers will not correctly account for the gas necessary for these callbacks upon estimation and may on accident or on purpose cause loss of funds in integrators.

Recommendation

Implement the `validateGasLeftForCallback` validation in the `afterGlvDepositExecution`, `afterGlvDepositCancellation`, `afterGlvWithdrawalExecution`, and `afterGlvWithdrawalCancellation` functions so keepers may adequately estimate gas and cannot accidentally or purposefully cause loss of funds in these systems.

Resolution

GMX Team: Resolved.

M-01 | Excess execution fee will be required

Category	Severity	Location	Status
Logical Error	Medium	GasUtils.sol: 343	Resolved

Description

To deposit GM, the long token address must be 0. `if (params.initialLongToken != address(0))`

However, when estimating execution fees, the only way to not pay for a deposit is to have the long address equal to the market address. `if (glvDeposit.market() == glvDeposit.initialLongToken())`

Due to these conflicting statements, users depositing GM tokens will be required to pay an execution fee as if they were depositing the underlying asset.

Recommendation

Consider only charging the deposit fee when an underlying long or short has been deposited.

Resolution

GMX Team: Resolved.

M-02 | GLV Withdrawal Callback Gas Cost Ignored

Category	Severity	Location	Status
Logical Error	Medium	GlvWithdrawalUtils: 207	Resolved

Description

For the GLV Withdrawal flow, the callback call is made at the end of `GlvWithdrawalUtils::executeGlvWithdrawal()`.

Since the callback is made after `payExecutionFee()` is called, the gas used in the callback will not come out of the user's execution fee and will instead be charged to GMX.

The max callback gas limit is set to 3,000,000 for Arbitrum and 2,000,000 for Avalanche, which over time will cause GMX incur substantial losses.

Recommendation

Move the callback call to before `payExecutionFee()` is called.

Resolution

GMX Team: Resolved.

M-03 | executionFee Should Be Updated

Category	Severity	Location	Status
Logical Error	Medium	GlvWithdrawalUtils.sol	Resolved

Description

The `createGlvWithdrawal` function does not update the `params.executionFee` after recording the transferred `wntAmount` with `recordTransferIn`. As a consequence, users may not receive a full refund when the transferred `wnt` amount exceeds the provided input value.

Furthermore, the `validateExecutionFee` function utilizes the provided `params.executionFee` instead of the actual transferred `wnt` amount, potentially causing the function to inaccurately revert even when the transferred amount is enough to cover execution costs.

Recommendation

Update the `params.executionFee` after recording the transfer.

Resolution

GMX Team: Resolved.

M-04 | GLV Used To Exit Illiquid Markets

Category	Severity	Location	Status
Logical Error	Medium	Global	Acknowledged

Description

GLV allows users to essentially swap between GM markets with no fees by triggering a GLV deposit of GM market A and triggering a GLV withdrawal of GM market B.

An actor holding GM market A can observe that GM market A is locked due to `pnlToPoolRatio` validations or reserves validations and use GLV to exit their GM A tokens into GM B tokens.

This will come at the expense of all other GLV holders, who are now left with the illiquid GM A tokens.

Recommendation

Consider applying an additional fee to GLV withdrawals to disincentivize this or disallowing GLV deposits when the underlying GM market is illiquid.

Resolution

GMX Team: Acknowledged.

M-05 | GLV Actions Cannot Be Simulated

Category	Severity	Location	Status
Logical Error	Medium	GlvHandler.sol: 232, 241	Resolved

Description

The `simulateExecuteGlvDeposit` and `simulateExecuteGlvWithdrawal` functions have the `onlyController` modifier, however they are not exposed through the `GlvRouter` and therefore cannot be called.

Recommendation

Expose the `simulateExecuteGlvDeposit` and `simulateExecuteGlvWithdrawal` functions through the `GlvRouter`.

Resolution

GMX Team: Resolved.

M-06 | GLV Used For Atomic Withdrawals

Category	Severity	Location	Status
Gaming	Medium	GlvWithdrawalUtils.sol: 247	Resolved

Description

In the _processMarketWithdrawal function the collected glv amount is converted into a market token amount and withdrawn from the GLV address.

However because the marketTokenAmount is determined during the time of execution, a user may abuse the GLV withdrawal to perform an atomic withdrawal while avoiding the atomic withdrawal fee.

Consider the following scenario:

- address(1) holds 1 GLV

- User A holds 1000 GLV

- The totalSupply of GLV is 1001

- User A creates a withdrawal for their 1000 GLV

- User A watches for the keeper's execution transaction and frontruns it to donate 50,000 GM tokens to the GLV in the same block

- The keeper's execution transaction now credits User A with 1000/1001 * 50,000 of these GM tokens for their GLV withdrawal

- User A is able to withdraw their GM tokens in a single block while only paying the TwoStep swapPricing fee and 10 basis points for their loss to the initial deposit address

Recommendation

This manipulation is not attractive when there are multiple even holders of a GLV supply, as the donated tokens will be split up amongst each holder. However, be wary of this gaming when launching new GLVs and consider using an internal storage for the tracking of the GM token balance of each GLV rather than relying on the balanceOf.

Resolution

GMX Team: Resolved.

M-07 | GLV Trapped Funds Upon Disabled

Category	Severity	Location	Status
DoS	Medium	GlvWithdrawalUtils.sol: 82	Resolved

Description

In the createGlvWithdrawal function the validateGlvMarket validation uses true as the shouldBeEnabled configuration. Therefore users may not withdraw from the portion of a Glv that is a disabled Glv market.

As a result when a market becomes disabled via the isGlvMarketDisabledKey key, users are not able to fully withdraw their deposited value from the Glv unless the keeper explicitly shifts all funds out of the disabled Glv market.

Recommendation

Consider allowing users to withdraw a disabled glv market, but not deposit it.

Resolution

GMX Team: Resolved.

M-08 | GLV Read-only Reentrancy Risk

Category	Severity	Location	Status
Reentrancy	Medium	GlvWithdrawalUtils.sol: 161	Resolved

Description

In the executeGlvWithdrawal function the withdrawal is performed with _processMarketWithdrawal before burning the glvWithdrawal.glvTokenAmount() from the glvVault. As a result any withdrawals which use shouldUnwrapNative token as true and receive weth will have the opportunity to exploit any systems which attempt to read the value of a GLV.

This is because during this transfer of native tokens to the receiver address the GLV supply has not yet been reduced, but the amount of GM tokens in GLV has been reduced. The receiver then gains control over the transaction execution if it is a contract with a receive function.

For example, any protocols attempting to use GLV as collateral can errantly count this collateral as being insufficient and allow incorrect liquidations since the GLV price is incorrectly reduced during this external transfer of native tokens.

Recommendation

Burn the GLV tokens before executing the withdrawal with the ExecuteWithdrawalUtils.executeWithdrawal function, but after the marketTokenAmount is computed with the _getMarketTokenAmount function.

Resolution

GMX Team: Resolved.

M-09 | User GLV Deposits Errantly Maximized

Category	Severity	Location	Status
Logical Error	Medium	GlvDepositUtils.sol: 360	Resolved

Description

In the `_getMintAmount` function the `poolValue` used to compute the value of the depositor's GM tokens is maximized to avoid applying a double spread from the GLV value computation. However the computation of the GLV value is not guaranteed to have a maximizing effect in excess of the maximizing effect experienced by the depositor.

For example:
• GLV is supports Markets A, B, and C
• GLV totalSupply is 100
• GLV holds 10 GM A, 10 GM B, and 0 GM C
• The minimum price of GM A is $1 and the maximum is $1.01
• The minimum price of GM B is $1 and the maximum is $1.01
• The minimum price of GM C is $1 and the maximum is $1.05 due to `indexToken` spread, trader pnl, impact pool amount, etc…
• User A deposits 10 GM C tokens
• The GLV value if minimized is $20, User A's deposits is $10 if minimized, User A would receive 50 GLV tokens if both were minimized
• The GLV value since it is maximized is $20.20, User A's deposit is valued at $10.50 since it is maximized
• As a result the maximization of both of these values positively affects User A such that they now receive $10.50/$20.20 * 100 ~= 51.98 GLV

Due to the maximization of both values in this example, User A receives roughly 2 more GLV tokens than if the values were to not be maximized. Additionally, the same effect would apply if a user is simply depositing more $ value than the GLV currently holds, since the maximization would have a greater absolute value impact on the numerator than the denominator due to being applied for a larger size.

Recommendation

Consider valuing the GLV value at the maximum, while valuing the user's deposits at the minimum to ensure that under no circumstances the protocol is rounding in the user's favor. Additionally, consider applying the same spread to withdrawals to protect the protocols from these cases. This behavior will negatively impact users, but protect against profitable arbitrages.

Resolution

GMX Team: Resolved.

M-10 | Virtual Inventory Ignored On Withdrawal

Category	Severity	Location	Status
Logical Error	Medium	Global	Acknowledged

Description

When withdrawing from a GM market in GMX V2, no price impact is applied since the withdrawn tokens are taken at the ratio of pool balances of the backing liquidity for the market. However if the market is within a virtual inventory, this ratio which was withdrawn directly from the market may have a negative impact on the virtual inventory (VI), but is ignored since no price impact is applied on withdrawal.

Consider the following scenario:
• Markets A, B, and C are all backed by WETH/USDC and make up a VI
• Market A holds $100 of WETH and $200 of USDC
• Market B holds $200 of WETH and $100 of USDC
• Market C holds $100 of WETH and $100 of USDC
• The total VI balances are $400 WETH & $400 USDC, the VI is balanced
• User A withdraws 50% of the GM supply from Market A
• Market A now holds $50 of WETH and $100 of USDC, ignoring fees
• The total VI balances are $350 WETH and $300 USDC, the VI is now unbalanced

Since there is no way to be positively impacted for balancing the virtual inventory, no value can be directly extracted this way. Though this leads to cases where the virtual inventory is negatively imbalanced, affecting other users who experience subsequent negative impact within the same virtual inventory on swaps and deposits, but while failing to negatively impact the user who created the imbalance.

Additionally, as the virtual inventory was implemented to disincentivize profitable manipulations in price impact across similar markets, ignoring the VI on withdrawals may allow a case where this is not sufficiently protected against, though none have been identified at this time.

Recommendation

Consider consulting the virtual inventory and applying the relevant negative impact if a withdrawal would cause an unwanted imbalance.

Resolution

GMX Team: Acknowledged.

M-11 | GLV Oracle Count Wrong For Homogenous Markets

Category	Severity	Location	Status
Unexpected Behavior	Medium	GasUtils.sol: 248	Acknowledged

Description

The `estimateGlvDepositOraclePriceCount` and `estimateGlvWithdrawalOraclePriceCount` functions add 2 constant oracle prices for every GLV. Additionally, the `estimateGlvShiftOraclePriceCount` is a static 4 oracle price feeds. However in the future it is planned that homogenous market GLV's may be supported.

Therefore the oracle price count is consistently over-estimated for GLV's with homogenous markets. As a result users and shifts will unintentionally have to pay a higher margin to the keepers on every interaction with a homogenous market GLV.

Recommendation

Consider adjusting the constant oracle price count for homogenous market GLVs to avoid charging excessive execution fees.

Resolution

GMX Team: Acknowledged.

M-12 | Insufficient Gas Forwarded For Refund

Category	Severity	Location	Status
Validation	Medium	CallbackUtils.sol: 89	Resolved

Description

In the `refundExecutionFee` function the `REFUND_EXECUTION_FEE_GAS_LIMIT` is forwarded to the callbackContract when calling the refundExecutionFee value. However the transaction execution is not validated to have the necessary `REFUND_EXECUTION_FEE_GAS_LIMIT` with the gas left for the transaction.

As the `REFUND_EXECUTION_FEE_GAS_LIMIT` is currently configured to 200,000 gas units and the refundExecutionFee call takes place at the end of action executions it is possible that keepers would not have provided enough gas to forward the entire limit. Instead 63/64 of the remaining gas left will be forwarded to the `refundExecutionFee` function call in these cases.

This may cause unexpected reverts for integrating systems and result in mis-accounted funds. However this instance of not validating the gas left with the `validateGasLeftForCallback` is not as severe as the lack of this validation for the GLV callbacks, as the refund fee is configured with a lower gas limit and handles less funds.

Recommendation

Validate the gas left for the `refundExecutionFee` callback with the `validateGasLeftForCallback` function.

Resolution

GMX Team: Resolved.

L-01 | Execution Fees Increased By Empty Markets

Category	Severity	Location	Status
Logical Error	Low	GlvHandler.sol: 320	Resolved

Description

The `payExecutionFee` function has a parameter `oraclePriceCountthat` is calculated as follows: `2 + marketCount + swapCount`. If there are no swaps involved, `oraclePriceCount` is equal to `2 + marketCount`.

In case there is GLV with one GM market, and a new GM market is added, `oraclePriceCount` moves from 3 to 4. This value is a multiplier used for `baseGasLimit`, when calculating `executionFeeForKeeper`.

As there is only a way to add markets, but not remove them, `marketCount` will monotonically increase. Disabling a market in glv is possible, but glv value will still iterate through all markets added.

Therefore, users will continuously pay more execution fees as `marketCount` increases, even if there are empty markets due to new ones added, or liquidity shifting, leaving markets empty.

Recommendation

Consider only counting non-zero value markets for the `oraclePriceCount`. Additionally, add a `onlyConfigKeeper` function to remove empty markets from the list.

Resolution

GMX Team: Resolved.

L-02 | Feature validation occurs before try-catch

Category	Severity	Location	Status
Logical Error	Low	GlvHandler.sol: 43	Resolved

Description

The GLV functions validate whether a feature is enabled before entering the try-catch block. This means that if the feature is not enabled, the execution will revert instead of cancelling the order.

This method of handling orders differs from the rest of GMX. Not cancelling an order when a feature is disabled means that users will not have access to their funds for a period of time. To manually cancel, users must wait for a predetermined amount of time.

Recommendation

To address this issue, it is advisable to move the feature validation check inside the try-catch block, similar to how it is implemented in other functions for both GLV deposits and withdrawals.

Resolution

GMX Team: Resolved.

L-03 | DoS Via Type Casting

Category	Severity	Location	Status
DoS	Low	GlvUtils: 128	Acknowledged

Description

`_getGlvMarketValue()` calls `getPoolValueInfo()` to determine the pool value for each market. The issue is that `_getGlvMarketValue()` will cast pool value to a `uint256`. However, pool value can potentially be a negative value if the PnL and impact pool are large enough. `_getGlvMarketValue()` is called in a loop inside of `getGlvValue()`, which, in turn, is called in the GLV deposit, withdrawal, and shift flow.

This means that if a single market in the GLV market has a negative pool value, any GLV feature will be disabled until that market has a positive pool value again. In practice, this is unlikely to occur because of ADL, `pnlToPoolRatio`, and reserve validations

Recommendation

Do not cast pool value to a `uint256`. This will allow for a more accurate calculation of the true value of the GLV market.

Resolution

GMX Team: Acknowledged.

L-04 | Unused marketCount Variable

Category	Severity	Location	Status
Superfluous Code	Low	GlvUtils.sol: 78	Resolved

Description

The `GlvUtils` contract contains two distinct `getGlvValue` functions. One of these functions utilizes an oracle to retrieve all market addresses based on market count, and is used in multiple processes such as deposit and withdrawal.

The other `getGlvValue` function is specifically accessed through a reader contract, where both market addresses and token prices are provided by the function caller. This function does not rely on `marketCount` and instead uses user-provided market addresses. Additionally, the input values are not validated within this function.

Recommendation

If the function is intended to be accessed via `GlvReader` without input validation, it is suggested to remove the lines `cache.marketListKey = Keys.glvSupportedMarketListKey(glv)` and `cache.marketCount = dataStore.getAddressCount(cache.marketListKey)` from the function as they are unnecessary.

Alternatively, if input validation is required, consider validating the user-provided `marketAddresses` array against `marketCount`.

Resolution

GMX Team: Resolved.

L-05 | GLV Deposit Config Added Twice

Category	Severity	Location	Status
Superfluous Code	Low	Config: 453-455	Resolved

Description

In `Config.sol`,`CREATE_GLV_DEPOSIT_FEATURE_DISABLED`, `CANCEL_GLV_WITHDRAWAL_FEATURE_DISABLED`, and `EXECUTE_GLV_DEPOSIT_FEATURE_DISABLED` are all assigned twice.

This is an unnecessary operation that wastes gas, and can be removed.

Recommendation

Remove the extra assignment from the `Config.sol`.

Resolution

GMX Team: Resolved.

L-06 | GM Deposits With False isMarketTokenDeposit

Category	Severity	Location	Status
Validation	Low	GlvDepositUtils.sol	Acknowledged

Description

The `createGlvDeposit` function includes an `isMarketTokenDeposit` parameter to specify if the deposited tokens are GM tokens. However, it is still possible to create a market token deposit request when `isMarketTokenDeposit` is false. This is due to the absence of any validation ensuring that the market token address is not the same as the initial long token or short token addresses.

The impact is minimal as the execution of a deposit will revert as the long and short tokens do not match, however invalid deposits should not be created.

Recommendation

Consider implementing checks to verify that `params.market != params.initialLongToken` and `params.market != params.initialShortToken` when `isMarketTokenDeposit` is `false`.

Resolution

GMX Team: Acknowledged.

L-07 | No Keeper Incentives To Create Shifts

Category	Severity	Location	Status
Logical Error	Low	GlvHandler.sol: 39	Acknowledged

Description

Keepers are in charge of creating and executing GLV shifts, to rebalance liquidity amongst GM pools. The issue relies on the shift creation, as there is no incentive for keeper to trigger this transaction.

First, any keeper that calls `createGlvShift` will need to pay execution fees in `wnt` tokens. Then, both `executeGlvShift` and `cancelGlvShift`, set uses keeper as `msg.sender` which is then used as the `refundReceiver` for the execution fee paid.

As an example:
• `KeeperA` creates a shift, pays execution fees
• `KeeperB` executes shift (or fails and cancels shift)
• `KeeperB` receives both the execution fees and the refund.
• `KeeperA` loses funds.

Recommendation

Consider storing the keeper address that triggers the shift creations, and set it as the refunds receiver for shift executions and cancellations.

Resolution

GMX Team: Acknowledged.

L-08 | Unused Struct Params

Category	Severity	Location	Status
Superfluous Code	Low	GlvShiftUtils.sol: 34	Acknowledged

Description

There are some struct declarations that contain parameters that are not in use:

- `CreateGlvShiftCache`;
- `fromMarket`, `toMarket`; and
- `toMarketTokenPrice`

Recommendation

Remove unused params in structs mentioned above.

Resolution

GMX Team: Acknowledged.

L-09 | Shifting Is Allowed To Same Market

Category	Severity	Location	Status
Validation	Low	GlvShiftUtils.sol: 71	Acknowledged

Description

Keepers have the ability to initiate a shift from one market to another. However, there is a missing validation to ensure that both markets are not the same.

Despite the missing validation, the shift execution will still be successful, resulting in withdrawals and deposits being made to the same market.

Recommendation

Implement a check to revert the transaction if both the `fromMarket` and `toMarket` are the same.

Resolution

GMX Team: Acknowledged.

L-10 | Cancelling Deposits Unnecessarily Transfers Tokens

Category	Severity	Location	Status
Optimization	Low	GlvDepositUtils.sol: 434	Resolved

Description

Users can opt to cancel their deposits, and receive their tokens back. The `cancelGlvDeposit` unnecessarily checks a non-zero value for all 3 token amounts (`marketToken`, `longToken`, `shortToken`), and transfers out these amounts to the account.

This is unnecessary as deposits will either have short and long tokens or market tokens, and never both.

Recommendation

Consider creating an `if-else` condition, so that long and short token amounts are not checked if `marketTokenAmount` is non-zero.

Resolution

GMX Team: Resolved.

L-11 | Malicious Code Can Be Set In Token Name/Symbol

Category	Severity	Location	Status
XSS	Low	GlvFactory.sol: 30	Resolved

Description

The `GlvFactory` allows `marketKeeper` to deploy a custom `glvToken` passing a name and symbol for the new token. Although currently this is a trusted origin, if the plan is to allow permissionless glv deployments, it is possible for an attacker to craft a name or symbol such that it includes markup that can contain Javascript code.

If loaded into a frontend without XSS protection, this can cause potential harm to users of the platform as was the case in the `EtherDelta` exploit. For more details on the `EtherDelta` exploit please refer to this article.

Recommendation

Sanitize or limit the length of the token `name` and `symbol` passed into the `createGlv` function from the `GlvFactory` contract.

Resolution

GMX Team: Resolved.

L-12 | Attacker can extract value from shifts

Category	Severity	Location	Status
Warning	Low	config.sol 17	Acknowledged

Description

An attacker can monitor the GLV keeper and when it identifies a shift being created, it can execute a swap on the `tomarket` to alter the pool's long/short ratio. This manipulation will result in a greater price impact than anticipated for the keepers shift, allowing the attacker to swap back into the pool the amount just swapped, thereby rebalancing the pool and capturing some value from the keeper's shift.

Although this extraction is limited by the price impact validation during GLV shifts, if a malicious actor can discern the logic used by a keeper to determine shifts they could repeatedly frontrun and extract value from the GLV this way.

Recommendation

Be aware of this potential manipulation and consider implementing safety rails in the keeper logic. Additionally, carefully consider this scenario when assigning the fee rates and slippage tolerance for shifts.

Resolution

GMX Team: Acknowledged.

L-13 | Missing function for feature key

Category	Severity	Location	Status
Logical Error	Low	Config.sol: 450	Resolved

Description

The CANCEL_GLV_SHIFT_FEATURE_DISABLED determines whether the glv shift cancellation feature is enabled or disabled. However, this key is currently unused as there is no cancel function for GLV shifts.

Recommendation

Implement a cancellation function for GLV or remove this key.

Resolution

GMX Team: Resolved.

L-14 | Unused errors

Category	Severity	Location	Status
Superfluous Code	Low	Errors.sol	Resolved

Description

The following errors have been created but are currently unused:

- `error EmptyGlvShift();`
- `error GlvInvalidReceiver(address glv, address receiver);`
- `error GlvInvalidCallbackContract(address glvHandler, address callbackContract);` and
- `error InvalidMarketTokenPrice(address market, int256 price)`

Recommendation

Consider implementing or removing the unused errors.

Resolution

GMX Team: Resolved.

L-15 | GLV Deposit Creation DoS

Category	Severity	Location	Status
DoS	Low	GasUtils.sol: 363	Acknowledged

Description

In the `estimateExecuteGlvDepositGasLimit` function the glv deposit gas limit is dependent on whether the deposit uses both the `initialLongTokenAmount` and the `initialShortToken` amount.

Therefore a user or integrator may provide an `executionFee` which is sufficient for a single token deposit while a malicious actor donates a single wei of `initialShortTokenAmount` to cause the deposit to have a higher estimated gas limit.

As a result the `executionFee` validation would fail if the user or integrator did not anticipate this and provide more than the necessary `executionFee` upon creation.

Recommendation

Document this for users and integrators so they can be sure to set an appropriate `executionFee` when DoS manipulations would have negative consequences.

Resolution

GMX Team: Acknowledged.

L-16 | GLV GM Balance Validation Minimizes Value

Category	Severity	Location	Status
Validation	Low	GlvShiftUtils.sol: 208	Resolved

Description

When validating that the GLV GM token balance is below the configured `maxMarketTokenBalanceUsd` amount, the pool value is minimized rather than maximized. As a result GM tokens which would exceed the `maxMarketTokenBalanceUsd` with maximize as true, but are less than this value when maximize is false are allowed.

Recommendation

Consider maximizing the pool value for to market during the `maxMarketTokenBalanceUsd` validation. And either use maximized for the from market price impact validation, or keep the pool value minimized for the to market price impact validation.

Resolution

GMX Team: Resolved.

L-17 | Users Charged For Oracles On Cancel

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

When a user cancels a deposit or withdrawal, there is no use of oracles.
However, the user is still charged for oracle usage because `oraclePriceCount` is calculated and passed to `payExecutionFee`. This will lead to users having to pay unnecessary execution fees.

These execution fees will be sent back to the account as they are the keeper in this scenario, however this unexpended gas amount should technically be delivered to the `refundReceiver` or `callbackContract` as an execution fee refund.

Recommendation

Pass `0` for the `oraclePriceCount` when calling `payExecutionFee` on a cancel.

Resolution

GMX Team: Acknowledged.

L-18 | Funds Are Not Synced In The GLV Vault

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

During GLV withdrawals and GLV shifts GM tokens are transferred out of the GLV contract and into the `glvVault`, however the balances for the glvVault are not synced to account for these additional tokens being added.

If a malicious actor would be able to trigger any action relying on a `recordTransferIn`, it would account for these GM tokens being credited to the attacker.
Currently there is no pathway for such an attack to occur, as the tokens are subsequently burned from the `glvVault` and the balance is synced.

However out of an abundance of caution it may be worthwhile to sync the token balance in the `glvVault` after transferring GM tokens from GLV.

Recommendation

Consider adding a `syncTokenBalance` function call to the GLV withdrawal and shift executions directly after transferring GM tokens out of glv.

Resolution

GMX Team: Resolved.

L-19 | GLV Deposits Incorrectly Value Fees

Category	Severity	Location	Status
Logical Error	Low	Global	Resolved

Description

During deposits the GLV value is computed based upon the value of the market tokens before the deposit occurs. However the deposit itself will increase the price of a GM token as fees are taken from the depositor and allocated to the pool. When determining the value of a user's received market tokens after the deposit, the deposit fees are included in that valuation.

This creates a small increase in the GM token valuation for the user's deposit relative to the GLV holdings. Additionally deposits which swap through the target market and charge more fees increase this effect. The user would be paying more in fees than the value of their GM tokens would increase relative to the GLV valuation, however this may have some non-trivial effect over longer periods with larger amounts of capital.

Additionally if a user were to currently hold a significant portion of the backing GM market liquidity then they would not be losing the fees which were allocated to the pool, and could potentially extract value from GLV this way, though this is an unlikely scenario.

Recommendation

Consider refactoring the GLV value computation to be performed after the deposit, but excluding the balance of GM tokens received from the deposit itself. An immediate fix may not be necessary, but it is worth being aware of and monitoring.

Resolution

GMX Team: Resolved.

L-20 | Incorrect Param Used in Event

Category	Severity	Location	Status
Logical Error	Low	GlvDepositUtils.sol: 275	Resolved

Description

The `emitGlvDepositExecuted` function emit an event that expects the last param to be `receivedMarketTokens` but `cache.mintAmount` is passed instead. Therefore, glv minted amount is emitted instead of GM tokens deposited by the user.

Recommendation

Update the last param passed in the function to `cache.receivedMarketTokens` instead of `cache.mintAmount`

Resolution

GMX Team: Resolved.

L-21 | Superfluous receivedUsd Value

Category	Severity	Location	Status
Optimization	Low	GlvDepositUtils.sol: 236	Resolved

Description

When calculating the minted value of a GLV deposit the `cache.receivedUsd` is assigned however not used afterwards.

Recommendation

Consider removing this value or implementing it's intended use-case.

Resolution

GMX Team: Resolved.

L-22 | Superfluous Imports

Category	Severity	Location	Status
Optimization	Low	ShiftHandler.sol: 8	Resolved

Description

The `ShiftUtils.sol` file is imported twice in the `ShiftHandler.sol` contract.

Recommendation

Remove the duplicated import.

Resolution

GMX Team: Resolved.

L-23 | Lacking GlobalNonReentrant Modifier

Category	Severity	Location	Status
Reentrancy	Low	GlvHandler.sol: 320	Resolved

Description

In the `GlvHandler` contract the `addMarketToGlv` function does not have the `globalNonReentrant` and therefore may be potentially unexpectedly re-entered into by a compromised `onlyConfigKeeper`.

Recommendation

Consider adding the `globalNonReentrant` modifier to the `addMarketToGlv` function.

Resolution

GMX Team: Resolved.

L-25 | Errant GLV Keys

Category	Severity	Location	Status
Logical Error	Low	Keys.sol: 85-86	Resolved

Description

In the Keys library there are `GLV_LONG_TOKEN` and `GLV_SHORT_TOKEN` keys which differ from the `LONG_TOKEN` and `SHORT_TOKEN` keys defined in the `GlvStoreUtils` library. These `GLV_LONG_TOKEN` and `GLV_SHORT_TOKEN` keys are not used currently, however they should be removed to prevent future errant use-cases. Additionally, the `GLV_KEY` in the `GlvStoreUtils` library is unused.

Recommendation

Remove the unused `GLV_LONG_TOKEN` and `GLV_SHORT_TOKEN` keys from the Keys library and remove the `GLV_KEY` from the `GlvStoreUtils` library.

Resolution

GMX Team: Resolved.

L-26 | GLV Salt Lookup Not Cleared

Category	Severity	Location	Status
Logical Error	Low	GlvStoreUtils.sol: 54	Acknowledged

Description

Upon adding a GLV with the `GlvStoreUtils.set` function, the GLV is given a lookup where the GLV address can be obtained with the GLV salt.

This uses the `getGlvSaltHash` result as the key in the dataStore, however this key is not cleared in the `GlvStoreUtils.remove` function.

Recommendation

Clear the `getGlvSaltHash` lookup from the `dataStore` in the `GlvStoreUtils.remove` function.

Resolution

GMX Team: Acknowledged.

L-27 | Misleading setMarketTokenAmount Function

Category	Severity	Location	Status
Typo	Low	GlvWithdrawalStoreUtils.sol: 126	Resolved

Description

The `GlvWithdrawalStoreUtils.setMarketTokenAmount` function sets the GLV token amount on a GLV withdrawal. This naming does not agree with what the function assigns.

Recommendation

Rename the function to `setGlvTokenAmount`.

Resolution

GMX Team: Resolved.

L-28 | Lacking GM Market Count Cap

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

In the GLV system there is no cap on the amount of GM markets which the trusted `configKeeper` can add to a GLV. In an extreme case this may cause very gassy deposit, withdrawal, and shift transactions which may require more gas than a single transaction can use.

Recommendation

Consider implementing a cap on the amount of GM markets which can be added to a GLV.

Resolution

GMX Team: Resolved.

L-29 | block.timestamp Usage

Category	Severity	Location	Status
Suggestion	Low	GlvShiftUtils.sol: 156	Resolved

Description

When setting the `glvShiftLastExecutedAtKey` value the `block.timestamp` is used. However this does not match the pattern of using the `Chain.currentTimestamp` function for the block timestamp.

Recommendation

Consider using `Chain.currentTimestamp` to follow the accepted pattern and automatically support any changes to the `currentTimestamp` function.

Resolution

GMX Team: Resolved.

L-30 | Users can imbalance glv to reduce glv yield

Category	Severity	Location	Status
Logical Error	Low	GlvHandler.sol: 250	Acknowledged

Description

The GLV shift feature can be exploited by temporarily increasing the utilization in a market that typically has low utilization. Once the keeper executes the shift, the attacker can lower the utilization back to its normal levels. Users can create this increase by opening a leveraged position briefly, consuming most of the available size to raise the utilization enough for a shift to occur. After the shift is completed, the attacker can close the position.

As a result, GLV holders will experience lower yield as they become stuck in a market with low utilization until another shift is performed. The attacker can then repeat this process, pressuring GLV holders to keep their funds in a less profitable market.

Recommendation

It is suggested to implement a time-weighted average utilization (TWAU) to prevent artificial spikes in utilization from triggering a keeper shift.

Resolution

GMX Team: Acknowledged.

L-31 | Withdrawal/Shift count reserved GM as available

Category	Severity	Location	Status
Logical Error	Low	GlvShiftUtils.sol: 90	Acknowledged

Description

When a keeper calls the createGlvShift function, it will perform the following check:
uint256 fromMarketTokenBalance = ERC20(params.fromMarket).balanceOf(params.glv);
The issue here is that a portion of the GLV's GM balance will be reserved by users who have created a withdrawal but have not been executed yet. Because of this, the shift will succeed even though it is moving funds that will soon be used for withdrawals.

This may result in the user's withdrawal failing because there is not enough GM token available for withdrawal. This can lead to wrongly failed withdrawals for users. Alternatively, if the withdrawal executes before the shift, the shift will fail.

Recommendation

It is recommended to establish a minimum percentage of GM tokens that should not be included in the shift, this check can be done off chain as the shifts will be performed by GMX keepers.

Additionally, the keeper is recommended to take into account the pending withdrawals when determining the shift amount.

Resolution

GMX Team: Acknowledged.

L-32 | Deposits With Zero Short Token Fail

Category	Severity	Location	Status
Logical Error	Low	GlvDepositUtils.sol: 133	Resolved

Description

In the `createGlvDeposit` function deposits with an `initialShortToken` are allowed to support markets where the long token is the same as the short token.

However in this case the `initialShortToken` should be the same token as the long token, not address(0).

Deposits with the `initialShortToken` as the zero address will revert upon depositing and expecting that the `outputToken` of the swap is the expected short token of the market.

https://github.com/gmx-io/gmx-synthetics/blob/1938e365dc009342aa288aa6b42fc1fd3cd9e45d/contracts/deposit/ExecuteDepositUtils.sol#L545

Recommendation

Require that neither the long token or short token are the zero address when creating a deposit.

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