Quantstamp Security Assessment Certificate

September 14th 2022 — Quantstamp Verified

GMX

This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	DEX
Auditors	Andy Lin, Senior Auditing Engineer Jennifer Wu, Auditing Engineer Roman Rohleder, Research Engineer
Timeline	2022-08-15 through 2022-09-07
EVM	Gray Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Medium

Source Code

Repository	Commit
gmx-io/gmx-contracts	787d767

Total Issues	**38** (3 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**1** (0 Resolved)
Low Risk Issues	**23** (2 Resolved)
Informational Risk Issues	**11** (1 Resolved)
Undetermined Risk Issues	**3** (0 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp performed an audit for the GMX project mainly on the deployed contracts and checked the difference against the latest `master` branch (commit: `787d767e033c4`) of the public Github repository to get the image of the quality. GMX is a decentralized spot and perpetual exchange based on pooled collateral and novel tokenomics.

The code base is quite complex, involving several layers of contract calls within the system. Unfortunately, the technical document does not explain the contract interactions and setups in detail. Aside from that, the responsibility separation of some contracts is ambiguous. For instance, the `PositionRouter.sol` and `PositionManager.sol` share similar functionalities, and the `OrderBook.sol` is not a typical order book with buy and sell orders waiting for matches. It is more of a contract to split the vault action into two steps. It was difficult to understand some of the logic - especially in `Vault.sol` due to the lack of code documentation. The code quality can be improved with more documentation, such as adding units (e.g., USD) for the variables. In addition, some issues were also identified in the September 2021 audit but unresolved. Testing-wise, the core part of the contracts are well covered (under `core/`) with branch coverage > 90%, while not all of the contracts are covered as thoroughly.

At the same time, this codebase is maintained by an anonymous group. Based on the GitHub repository activities, most of the development is led by 4 users: xvi10 (protocol account), gdev8317, xdev10, gkrasulya, and nissoh. Protocol account xvi10 contributes the majority of the codebase and development; it is unclear if the development is from an individual or a group. We also noted a lack of continuous integration/regression framework setup. Some pull requests are also merged/closed without a review from another developer, which is another concern/vulnerability. The development process can be improved with a better security/integration framework. We want to emphasize that new upgrades in the implementation or deployment errors can further introduce vulnerabilities in the protocol. A lack of robust input validation and continuous integration framework can only increase the likelihood and impact in the event of mistakes.

The previous audit report was published in September 2021, and the report reviewed commit `903531` for contracts: `Router.sol`, `Vault.sol`, `USDG.sol`, and `YieldToken.sol`. It is important to note that the audit report is not a full audit: only 4 files were audited, and other contracts such as `Timelock.sol` are omitted. For example, `Timelock.sol` is the governor contract used to interact with the `Vault.sol` and `GLPManager.sol` to facilitate actions such as withdrawing funds from the vault. It is unclear how many issues were resolved from the audit; it looks like only 4 major issues were resolved from the audit report's conclusion.

We recommend that the users keep the contract risk in mind and interact with the system conservatively due to its complexity. The complexity involves not only the code but also the operations. For the contracts to work correctly, the contracts have to be set with the correct configurations. Aside from that, the users should also be aware of the risk of the privileged roles. The admin of the `Timelock.sol` contract can set several things to the `Vault.sol` contract, including changing the governance contract. The users will only have a specific `buffer` period (currently set to 1 day) to react against it.

Table 1: Differences between Deployed and Github Repo Commit `787d767`

Contract	Deployed on Arbitrum vs. Github Repo Commit 787d767 (latest commit)
GMX.sol	No difference.
OrderBookReader.sol	No difference.
OrderBook.sol	Additional override on functions: `executeSwapOrder`, `executeIncreaseOrder`, `executeDecreaseOrder` in the deployed contract.
GlpManager.sol	Additional override on state variables.
PositionManager.sol	Difference in function call order `executeDecreaseOrder` function when enabling leverage before `getDecreaseOrder` function. The following functions: `enableLeverage`, `executeDecreaseOrder`, and `disableLeverage` functions are called before `getDecreaseOrder` in the deployed contract. In commit 787d767, the functions are executed **after** `getDecreaseOrder`. **The bug fix is noted in the issue below.**
PositionRouter.sol	No difference.
Reader.sol	No difference.
RewardRouterV2.sol	No difference.
RewardTracker.sol	New storage variable `totalDepositSupply` used in `_stake` and `_unstake` functions.
Router.sol	No difference.
StakedGlp.sol	No difference.
Vault.sol	**Significant Difference**; modified functions such as `_collectMarginFees` , and `updateCumulativeFundingRate`; new functions such as `_increaseGlobalShortSize`; added new state variables; additional verification in `increasePosition`; abstracted functionalities to **vaultUtils.sol**; bug fixes in functions `buyUSDG`, `liquidatePosition`, and `_collectSwapFees`. **The bug fixes are noted in the issues below.**
Timelock.sol	**Different**; additional functions, some modifier changes from `onlyAdmin` to `onlyAdminOrHandler`.

ID	Description	Severity	Status
QSP-1	Ability to Rug-Pull	^ Medium	Unresolved
QSP-2	Operational Risk	⌄ Low	Unresolved
QSP-3	Manipulate Vault's Value	⌄ Low	Unresolved
QSP-4	Positions Cannot Be Closed when Leverage Is Disabled	⌄ Low	Unresolved
QSP-5	Deployed Contracts Allowing 100x Leverage	⌄ Low	Unresolved
QSP-6	Higher Position Fee Charged when Leverage Is Disabled	⌄ Low	Unresolved
QSP-7	Unable to Remove a Token From Whitelist	⌄ Low	Unresolved
QSP-8	Missing Input Validation	⌄ Low	Unresolved
QSP-9	Use of Old Solidity Version (0.6.12)	⌄ Low	Unresolved
QSP-10	Uncapped Fee `minExecutionFee` and `minPurchaseTokenAmountUsd`	⌄ Low	Unresolved
QSP-11	Uncapped Fee `BasePositionManager.depositFee`	⌄ Low	Unresolved
QSP-12	`increasePositionBufferBps` Overflowable, Leading to Fees Being Always Collected	⌄ Low	Unresolved
QSP-13	Uncapped Position Sizes in `BasePositionManager.setMaxGlobalSizes()`	⌄ Low	Unresolved
QSP-14	Application Monitoring Can Be Improved by Emitting More Events	⌄ Low	Unresolved
QSP-15	Denial of Service Due to Unbound Iteration	⌄ Low	Unresolved
QSP-16	Outdated Value Causing Risk of Wrong Incentive for GLP Weight Rebalancing	⌄ Low	Unresolved
QSP-17	Front-Run Orderbook Execution Functions	⌄ Low	Unresolved
QSP-18	Front-Run Vault Actions	⌄ Low	Unresolved
QSP-19	Ignoring Return Value of ERC-20 Transfer	⌄ Low	Unresolved
QSP-20	Accessing Wrong Price Feed Data	⌄ Low	Unresolved
QSP-21	USDG Accounting Mismatch with Total Supply	⌄ Low	Unresolved
QSP-22	Risk of Inaccurate GLP Token Minting	⌄ Low	Acknowledged
QSP-23	AMM Price Excluded From Pricing Feed	⌄ Low	Fixed
QSP-24	Missing Position Data	⌄ Low	Fixed
QSP-25	Allowance Double-Spend Exploit	○ Informational	Unresolved
QSP-26	Missing Implementation of Vault Access Function in Timelock	○ Informational	Unresolved
QSP-27	Functions Available While Uninitialized	○ Informational	Unresolved
QSP-28	Missing Upgrade Path for Vault Implementation	○ Informational	Unresolved
QSP-29	Critical Role Transfers Not Following Two-Step Pattern	○ Informational	Unresolved
QSP-30	Unlocked Pragma	○ Informational	Unresolved
QSP-31	Reentrancy Risk	○ Informational	Unresolved
QSP-32	Concerning Unexecuted Time-Locked Actions	○ Informational	Unresolved
QSP-33	Risk of Self Collateralization	○ Informational	Unresolved

ID	Description	Severity	Status
QSP-34	Swap Pricing Disabled	○ Informational	Fixed
QSP-35	Privileged Roles and Ownership	○ Informational	Unresolved
QSP-36	Stop Users From Redeeming GLP	? Undetermined	Unresolved
QSP-37	`BaseToken.sol` Allows Changing `_name` and `_symbol` After Deployment	? Undetermined	Unresolved
QSP-38	Unclear Intentions and Specs	? Undetermined	Unresolved

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

DISCLAIMER:
The scope of this audit is limited to the contracts in the deployed contracts page: `BaseToken.sol`, `BasePositionManager.sol`, `GlpManager.sol`, `GMX.sol`, `MintableBaseToken.sol`, `OrderBook.sol`, `OrderBookReader.sol`, `PositionManager.sol`, `PositionRouter.sol`, `Reader.sol`, `RewardRouterV2.sol`, `RewardTracker.sol`, `Router.sol` , `StakedGlp.sol`, `Timelock.sol`, and `Vault.sol`. If we sense any issues from contracts out-of-scope, we might add them to the report. But we did not fully review the contracts outside of the scope. Also, this audit is not directly requested by the development team. Thus, the path to fixing the issues is unclear to us. Lastly, there are still rapid changes in the `master` branch. We chose the latest commit (`787d767e033c4`) during our review. However, our main focus is on the deployed contracts and mainly uses the `master` branch code for reference as it is still under rapid development.

Also, if the final commit hash provided by the client contains features that are not in the scope of the audit or our fix review, those features are excluded from the consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Ability to Rug-Pull

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `BaseToken.sol`, `MintableBaseToken.sol`, `Timelock.sol`, `Vault.sol`

Description: Some contracts have privileged roles that can have enormous power against the user. Also, it seems like an anonymous team operates the GMX, so the users should carefully consider this risk. The following is the list of places where the admin or the operation team can potentially take away the funds from the user:

1. `Timelock.sol`: the `Timelock` contract is designed to be the "governance" of the `Vault` contract. However, the "admin" of the `Timelock` contract can easily set several configurations for the `Vault` without waiting for the buffer period of the `Timelock` (e.g., `setMaxLeverage`, `setFundingRate`, `setFees`...e.t.c). Also, the `setGov` function will allow the admin to swap the `Timelock` contract to a potentially malicious one after the `buffer` period (currently set to 1 day, max to 5 days) and then have access to all privileged functions in the `Vault.sol` that holds all the funds.
2. `BaseToken.sol`: once the governance contract approves the "handler" role, the "handler" can transfer on behalf of the user to transfer the user's token away. The issue applies to the `GLP`, `EsGMX`, and `GMX` tokens.
3. `MintableBaseToken.sol`: the "minter" role will have the ability to not only mint but also burn any user's token. The issue applies to the `GLP`, `EsGMX`, and `GMX` tokens.

Exploit Scenario: The following is a sample potential scenario if the admin of the `GMX` decides to rug pull:

1. The admin first calls `Timelock.sol:signalSetGov`.
2. A mass exit is triggered. All users should close the position and withdraw from the `GMX`. However, there is only 1 day (`buffer`) to react, and the network might not be able to handle all exits within the time due to gas constraints.
3. After the `buffer` period, the admin calls `Timelock.sol:setGov` to swap the governance contract from the `Timelock.sol`.
4. Now, the admin can call the `Vault.sol:upgradeVault` to transfer all tokens away via the new malicious governance contract.

Recommendation: We recommend limiting those privileged roles to contracts with limited function and ability. Also, the team should provide detailed documentation around all privileged roles and the expected role actor.

QSP-2 Operational Risk

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Whole System`

Description: GMX has complicated contract setups with multiple layers of contract calls and interactions. The contract can only work as expected if configured correctly. Nonetheless, judging from the repository code base, it lacks post-deployment checks to ensure the configuration is as expected.
Meanwhile, tokenomics heavily relies on (manual) operations. For instance, the fee distribution requires the admin to call `Timelock.sol:withdrawFees` to take the fee from the vault and redistribute it to the correspondent `RewardDistributor.sol` contracts.

Exploit Scenario: The following are some potential scenarios to showcase the risk due to the complexity of the contract setups:

1. The `Vault.sol:isLeverageEnabled` is supposed to be `false` so that only the `PositionManager.sol` contract can call the vault to increase, decrease or liquidate a position. However, in the `Vault.sol` contract, the default value for the `isLeverageEnabled` is true. So right after the deployment, the flag is true. If the admin forgets to set the value to false, users can directly interact with the `Vault.sol` contract to start a position. This will bypass the validation logic in the `PositionManager.sol:_validateIncreaseOrder`.
2. The admin makes a mistake on the distribution of the fee. Instead of distributing 30% to staked GMX and 70% to staked GLP, the admin sends 70% of the fee to staked GMX instead.

Recommendation: We recommend that the team add post-deployment checks to check the public fields are set correctly and run a smoke and integration test after deployment to each environment. Long term, the team should consider embedding fee distribution logic into the smart contract.

QSP-3 Manipulate Vault's Value

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `GlpManager.sol`

Description: The calculated AUM value can be manipulated using `aumAddition` and `aumDeduction`. These values can be set using `GlpManager.setAumAdjustment` function which is only accessible by the governor contract. Although the current governor contract `Timelock` is missing the access function for `Glp.setAumAdjustment` function, the documentation should caution users that the Vault's AUM can be adjusted as there are no limits when applying `aumAddition` and `aumDeduction`. It is unclear why `aumAddition` and `aumDeduction` are needed. Since the value of the GLP token is determined by the Vault's AUM, the price of the GLP can be manipulated. Therefore, it is possible to prevent users from withdrawing assets by modifying the value of the Vault and therefore the GLP's value.

Exploit Scenario:

1. A new timelock contract is deployed with access to the function `GlpManager.setAumAdjustment`.
2. A malicious admin changes the value of `aumDeduction` to deflate the Vault's value.
3. `GlpManager.getAum` calculates lower AUM value.
4. Users withdraw fewer assets due to manipulated AUM value of the Vault.

Recommendation: Remove the usage of `aumDeduction` and `aumAddition` when calculating the Vault's AUM value.

QSP-4 Positions Cannot Be Closed when Leverage Is Disabled

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `PositionRouter.sol`

Description: If leverage is disabled in `PositionRouter`, it is impossible to use `PositionRouter` to close existing positions. When executing a decrease position using `PositionRouter.executeDecreasePosition`, the position is validated using `PositionRouter._validateExecution`. This function will revert if leverage is disabled. Leverage can be readily disabled and enabled using `Timelock.enableLeverage` and `Timelock.disableLeverage`. Note that there is no time delay when using these functions.
However, it is possible to close the existing position using the `Vault.decreasePosition` function. When leverage is disabled, a 1% fee will be applied to the position's change in the `getPosition` function. It is unclear if this is intentional based on the available documentation.

Recommendation: Update the documentation to inform users that existing positions cannot be decreased when leverage is disabled on the protocol. Update the documentation to specify what is allowed and what is not allowed when the protocol disables leverage.

QSP-5 Deployed Contracts Allowing 100x Leverage

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`, `Timelock.sol`

Description: In the deployed `Vault` (see: explorer), the `maxLeverage` is set to 1000000, which is 100x (1000000/10000, 10000 is the `BASIS_POINTS_DIVISOR`). The `maxLeverage` variable is used in `Vault.validateLiquidation` function to verify positions' leverages. Note that we did not identify additional maximum leverage verification in `PositionManager` and `PositionRouter` contracts. Thus, based on the value of the deployed contracts, the current maximum leverage is 100x.
This disobeys the statement on the home page of GMX where it states: "Trade BTC, ETH, AVAX and other top cryptocurrencies with up to 30x leverage directly from your wallet". Also, the proposal to increase the leverage to 50x seems not to have reached consensus yet despite the last comment being in 2022-05-05.

Update: Update the documentation to reflect the new maximum leverage limit or revert the maximum leverage back to 30x.

QSP-6 Higher Position Fee Charged when Leverage Is Disabled

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`, `Timelock.sol`

Description: Based on the GMX trading documentation, the position fee charged for opening a position or closing a position is 0.1% of the position size. However, when leverage is disabled, the position fee is set to 1%.
In `Vault.sol`, the `getPositionFee` function returns the position fee using `BASIS_POINTS_DIVISOR` and `marginFeeBasisPoints`.
When leverage is disabled by `Timelock.disableLeverage` function, the position fee is changed from `marginFeeBasisPoints` (0.1%) to `maxMarginFeeBasisPoints` (1%) in `Timelock`.
Although it is not possible to open or increase a position when leverage is disabled, an existing position can be decreased in `Vault.decreasePosition` external function. When leverage is disabled, a 1% fee will be applied to the position's change in the `getPosition` function.

Recommendation: Update the documentation to inform users that trading fees are increased to `maxMarginFeeBasisPoints` when closing a position while leverage is disabled.

QSP-7 Unable to Remove a Token From Whitelist

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`, `Timelock.sol`

Description: In the current implementation of `Vault.sol` and `Timelock.sol`, it is not possible to remove a token from `whitelistedTokens`. Once a token is added in `Vault.sol` through `setTokeConfig(...)` function, the mapping `whitelistedTokens` for the token is set to `true`. The token can only be removed from the whitelist if `Vault.sol:clearTokenConfig(...)` function is called, which is only accessible to the `gov` address. Since the `Timelock` contract (the `gov` of the `Vault.sol`) is missing an access function to use `clearTokenConfig(...)` function, it is not possible to remove the token from the whitelist.
This is problematic because `whitelistedTokens` is used to validate a transaction for the following functions in `Vault.sol`:

1. `buyUSDG(...)` function
2. `sellUSDG(...)` function
3. `swap(...)` function
4. `increasePosition(...)` function

Without the ability to remove whitelisted tokens, the GLP will not be able to react flexibly with rapid changes on a token situation.

Exploit Scenario: If a token is hacked or having regulation issues, the team might want to de-list the token. However, the team will not be able to do so unless swapping the `gov` of the `Vault`.

Recommendation:

1. Decouple the usage of `whiteListedToken` in `GlpManager.getAum`.
2. Add a function access to the function `clearTokenConfig` from the `Timelock.sol` contract.

QSP-8 Missing Input Validation

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `All contracts`

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.
Following is the list of places that can potentially benefit from a stricter input validation but is not limited to:

- `BaseToken.sol`: the `_gov` of the `setGov(...)` function should not be zero address. Once the `gov` address is set to zero address, it is no longer possible to access `onlyGov` functions, including the `setGov` function. `BaseToken` is inherited by `GMX` and `GLP`.
- `GlpManager.sol` : the `_cooldownDuration` of the `constructor` should be less than `MAX_COOLDOWN_DURATION` before setting `cooldownDuration`.
- `Governable.sol`: the `_gov` of the `setGov(...)` function should not be zero address. `Governable` is inherited by `GlpManager`.
- `StakedGlp.sol`: the `_glp` of the `constructor` should not be zero address.
- `StakedGlp.sol`: the `_stakedGlpTracker` of the `constructor` should not be zero address.
- `StakedGlp.sol`: the `_feeGlpTracker` of the `constructor` should not be zero address.
- `OrderBook.sol`: the `_triggerPrice` of the `updateIncreaseOrder` should be greater than zero.
- `OrderBook.sol`: the `_sizeDelta` of the `updateIncreaseOrder` should be greater than zero.
- `OrderBook.sol`: the `_feeReceiver` of the `executeSwapOrder` should not be zero address.
- `OrderBook.sol`: the `_feeReceiver` of the `executeIncreaseOrder` should not be zero address.
- `OrderBook.sol`: the `_feeReceiver` of the `executeDecreaseOrder` should not be zero address.
- `OrderBook.sol`: the `_path` array length of the `createIncreaseOrder` should not be zero.
- `OrderBookReader.sol`: the `_account` of the `getDecreaseOrders` should not be zero address.
- `OrderBookReader.sol`: the `_account` of the `getIncreaseOrders` should not be zero address.
- `OrderBookReader.sol`: the `_account` of the `getSwapOrders` should not be zero address.
- `Vault.sol`: the `_liquidationFeeUsd` of the `initialize` function should be less than or equal to `MAX_LIQUIDATION_FEE_USD`.
- `Vault.sol`: the `_fundingRateFactor` of the `initialize` function should be less than or equal to `MAX_FUNDING_RATE_FACTOR`.
- `Vault.sol`: the `setError` function does not check whether or not parameter `_errorCode` already points to an existing error code, allowing to overwrite existing error codes with different messages.
- `Vault.sol`: the `_receiver` of the `sellUSDG` should not be zero address.
- `Vault.sol`: the `_receiver` of the `sellUSDG` should not be zero address.
- `Vault.sol`: the `_receiver` of the `buyUSDG` should not be zero address.
- `Vault.sol`: the `_receiver` of the `swap` should not be zero address.
- `Vault.sol`: the `_receiver` of the `increasePosition` should not be zero address.
- `Vault.sol`: the `_receiver` of the `decreasePosition` should not be zero address.
- `Vault.sol`: the `_receiver` of the `liquidatePosition` should not be zero address.
- `PositionManager.sol`: the `_orderBook` of the constructor should not be zero address.
- `PositionRouter.sol`: the `_minTimeDelayPublic` of the `setDelayValues` function should be should be less than or equal to the `_maxTimeDelay` parameter.
- `Reader.sol`: the `_token` of the `getTotalBalance` should not be zero address. If zero address is accepted to process ether balance, then the function needs to be updated similarly to `getTokenBalances` function.
- `RewardRouterV2.sol`: the `_receiver` of the `unstakeAndRedeemGlpETH` should not be zero address.
- `RewardRouterV2.sol`: the `_receiver` of the `unstakeAndRedeemGlpETH` should not be zero address.
- `RewardRouterV2.sol`: the `distributor` of the `signalTransfer` should not be zero address.
- `RewardTracker.sol`: the `_receiver` of the `constructor` should not be zero address.
- `RewardTracker.sol`: the `_receiver` of the `_claim` should not be zero address.
- `Router.sol`: the `_vault`, `_usdg`, `_weth` of the `constructor` should not be zero addresses.
- `Router.sol`: the `_gov` of the `setGov` should not be zero address. Once the `gov` address is set to zero address, it is no longer possible to access `onlyGov` functions including `setGov` function.

Recommendation: Add the validations listed in the description.

QSP-9 Use of Old Solidity Version (0.6.12)

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `All Contract`

Description: As security standards develop, so does the Solidity language. To stay up to date with current practices, it is important to use a recent version of Solidity and recent conventions. Newer solidity versions not only fix (security) issues but also may introduce implicit security improvements (i.e. see the added implicit underflow/overflow checks introduced in solidity 0.8.0). Note: For example, unchecked overflow/underflows have been discovered and integrated into this report. Other occurrences (such as contracts out of scope for this audit) may still exist in live code. The previous ABDK audit report has also identified some overflow/underflow-related issues (CVF-88 and CVF-94). Those issues are still not fixed.

Recommendation: Consider updating all contracts to the latest major solidity version (0.8.*).

QSP-10 Uncapped Fee `minExecutionFee` and `minPurchaseTokenAmountUsd`

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `OrderBook.sol`, `PositionRouter.sol`

Description: Function `OrderBook.setMinExecutionFee()` does not constrain the input `_minExecutionFee`. Consequently, users can be subject to very high fees. This can subject users to high execution fees (order book swaps, order increases/decreases) without prior announcements or make the protocol unusable.
Further, function `OrderBook.setMinPurchaseTokenAmountUsd()` and the corresponding state variable `minPurchaseTokenAmountUsd` are impacted, as used to be checked against in order increasing function `createIncreaseOrder()`.
Note: Only the `gov` address can execute these functions. Similarly, `PositionRouter.setMinExecutionFee()` is impacted.

Recommendation: Consider having an upper bound for the execution fee, communicate it via public facing documentation and accordingly check against it in functions `setMinExecutionFee()` and `initialize()` (`constructor()` in the case of `PositionRouter.sol`). And a corresponding upper limit for `OrderBook.setMinPurchaseTokenAmountUsd()`. Same for the `PositionRouter.setMinExecutionFee()` function,

QSP-11 Uncapped Fee `BasePositionManager.depositFee`

Severity: *Low Risk*

Status: Unresolved

Description: Function `BasePositionManager.setDepositFee()` does not constrain the input `_depositFee`. Consequently, users can be subject to very high fees (effectively up to 100%, `BASIS_POINTS_DIVISOR = 10000`). This can subject users to high fees with no prior announcements.
Note: Only the `admin` address can execute this function.

Recommendation: Consider having an upper bound for the deposit fee, communicate it via public-facing documentation, and accordingly check against it in function `setDepositFee()`.

QSP-12 `increasePositionBufferBps` Overflowable, Leading to Fees Being Always Collected

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `BasePositionManager.sol`, `PositionManager.sol`

Description: Function `BasePositionManager.setIncreasePositionBufferBps()` does not constrain the input `_increasePositionBufferBps` in either direction. In combination with the use of the unsafe addition in function `_shouldDeductFee()`

```
uint256 nextLeverage = nextSize.mul(BASIS_POINTS_DIVISOR + increasePositionBufferBps).div(nextCollateral);
```

could lead to a multiplier smaller than `BASIS_POINTS_DIVISOR = 10000` and potentially always returning true for (`return nextLeverage < prevLeverage`) and lead to fees always being collected, regardless of leverage change.
Similarly in contract `PositionManager.sol`, if state variable `shouldValidateIncreaseOrder` is set to `true`, an overflowed `increasePositionBufferBps` value could lead to function `_validateIncreaseOrder()` always reverting, preventing the execution of order increasing function `executeIncreaseOrder()`. This is due to the same shared code between `BasePositionManager._shouldDeductFee()` and `PositionManager._validateIncreaseOrder()`, both of which containing the same overflow issue.
Note: Only the `admin` address can execute this function.

Recommendation: Consider having a sane upper and lower bound for `increasePositionBufferBps`, communicate it via public-facing documentation, and accordingly check against it in function `setIncreasePositionBufferBps()`. And/or consider replacing the unsafe addition operation in `_shouldDeductFee()` with its safe counterpart (`.add()`).

QSP-13 Uncapped Position Sizes in `BasePositionManager.setMaxGlobalSizes()`

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `BasePositionManager.sol`

Description: Function `BasePositionManager.setMaxGlobalSizes()` does not constrain the inputs `_longSizes[]` and `_shortSizes[]` to a sane lower bound, potentially making certain tokens unusable if set to economically very low values.
Note: Only the `admin` address can execute this function.

Recommendation: Consider having a lower bound for `_longSizes[]` and `_shortSizes[]`, communicate it via public facing documentation and accordingly check against it in function `setMaxGlobalSizes()`.

QSP-14 Application Monitoring Can Be Improved by Emitting More Events

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`, `BaseToken.sol`, `StakedGlp.sol`, `GlpManager.sol`

Description: To validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transition can be logged, which is beneficial for monitoring the contract and tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve the application management:

1. `Vault.sol`: `setManager()`, `setGov()`, `setMaxLeverage()`, ...
2. `BaseToken.sol`: `setGov()`, `setYieldTrackers()`, ...
3. `GlpManager.sol`: `setInPrivateMode()`, `setHandler()`, ...
4. `Router.sol`: `setGov()`, `addPlugin()`, `removePlugin()`, ...

In particular, non-standard ERC20 contract `StakedGlp.sol` does not emit `Transfer()` events when transferring tokens.

Recommendation: Consider emitting the events.

QSP-15 Denial of Service Due to Unbound Iteration

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `BaseToken.sol`, `Reader.sol`, `GlpManager.sol`, `Vault.sol`

Description: There is a limit on how much gas a block can execute on the network. It can consume more gas than the network limit when iterating over an unbounded list. In that case, the transaction will never work and block the service. The following is the list of places that are of risk:

1. In `BaseToken.sol`, the following functions: `recoverClaim`, `claim`, `_updateRewards` loop through the `yieldTrackers` array.
2. In `Reader.sol`, the function `getTokenSupply` loops through the `_excludedAccounts`. It is quite likely the `_excludedAccounts` will grow as the users grow, and the function `getTokenSupply` cannot work without giving the full list of the excluded accounts. Thus, the risk of this function eventually not working will grow as the system operates longer. Similar applies to the `getTotalBalance` function.
3. The `GlpManager.sol:getAum` functions loops through the `vault.allWhitelistedTokens`. However, the `Vault.sol:setTokenConfig` does not have a cap on the max amount of whitelisted tokens. If too many tokens are whitelisted, the `GlpManager.sol:getAum` function will block users from buying and selling GLP.

Recommendation:

1. In `BaseToken.sol`, set a cap for the `yieldTrackers` and validate it during the `setYieldTrackers` function.
2. Since both of the `Reader.sol:getTokenSupply` and `Reader.sol:getTotalBalance` is not called within the contracts, the impact is unclear. We recommend the team double-check this usage and ensure the `_excludedAccounts` would not grow infinitely.
3. Set a cap for the max amount of the whitelisted tokens in the `Vault.sol:setTokenConfig` function.

QSP-16 Outdated Value Causing Risk of Wrong Incentive for GLP Weight Rebalancing

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`

Description: The GLP is designed to auto-rebalance with the dynamic fee (see: doc). The fee will be less when the action moves toward the desired rate, and an extra fee will be charged when the opposite is the case. However, in the `Vault.sol:getFeeBasisPoints` function, whether the action is moving toward the desired rate is based on the `usdgAmounts[token]` mapping (the line `initialAmount = usdgAmounts[_token]`). The `usdgAmount` mapping caches the USD value when executing an action. However, if the token price fluctuates after the action timing, it will not reflect with the token price update. Consequently, the rebalancing can be inaccurate and, in the worse case, move toward the opposite direction on the intention of the dynamic fee as it does not reflect the real-time value of the tokens.
Note: The admin can call the `Timelock.sol:setUsdgAmount` function to force update the `usdgAmount` of any token.

Exploit Scenario:

1. For simplicity, assume the target weight of the GLP is 50% of ETH and 50% of USD.
2. At T0, the ETH price is 1000. There is 1 ETH and 1000 USD in the pool. The weight is perfect.
3. At T1, the ETH price drops to 500. The pool has a 500 USD equivalent of ETH and a 1000 USD equivalent in the pool.
4. Alice buys USDG at T1 with ETH. Ideally, the GLP should rebalance toward the same value of ETH and USD back again, so adding ETH to the pool to buy USDG should be encouraged with less fee. However, in the case here, since the `usdg[ETH] = 1000`, the contract will consider adding ETH as moving further away from the target weight and charge an extra fee instead.

Recommendation: We do not have a clear recommendation for now as it needs a more considered design change to fix this. However, the team should be aware of this and monitor the difference between the cached `usdg` value and the actual value. Users should be cautious of this behavior as well.

QSP-17 Front-Run Orderbook Execution Functions

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `OrderBook.sol`

Description: There are several "execution functions" in the `OrderBook.sol` contract. Those functions will give an `order.executionFee` to the `_feeReceiver` address from the input. However, an attacker can monitor the mempool, copy the transaction, and change the `_feeReceiver` to front-run and collect the `order.executionFee`. An attacker will be incentivized as long as the `order.executionFee` is larger than the gas cost. On the other hand, if the `order.executionFee` is lower than the gas cost, then the executor will need to bare an operation loss when calling the function. The following is the list of impacted functions: `executeSwapOrder`, `executeIncreaseOrder`, and `executeDecreaseOrder`

Exploit Scenario:

1. The team's executor sends the transaction to `executeSwapOrder` on the `OrderBook.sol` contract.
2. Alice sees the transaction. She copies the transaction and changes the `_feeReceiver` field to herself.
3. Alice front-run the transaction and gets the execution fee.
4. The team's executor will bear a small transaction gas lost as the transaction will fail.

Recommendation: A potential direction is that the order should specify a preferred executor. Only the preferred executor can run the execution for the first N minutes. However, anyone can execute once a certain period (N minutes) passes. Meanwhile, the team should monitor the gas costs of the "execution" functions and ensure the `order.executionFee` should be as close to the gas cost as possible to mitigate the issue.

QSP-18 Front-Run Vault Actions

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`

Description: The `Vault.sol:_transferIn` function uses the difference between the current balance and the previously cached balance data as the amount of the transferred token. The design allows an attacker to front-run vault actions requiring token transfer. If a user does not ensure the token transfer and the vault action call are atomic, the attacker can call the vault after the user transfers the token into the `Vault` contract.
The vulnerability is mitigated by the upper contracts that call the functions from the `Vault.sol` contract. For instance, the `Router.sol` contract does the token transfer and the vault action together. As a result, the users will not be impacted if they do not directly interact with the `Vault.sol` contract.

Exploit Scenario:

1. Alice wants to swap 10 WETH for USDG.
2. Alice first sends a transaction to transfer 10 WETH to the `Vault`.
3. Bob monitors the transaction and calls the `swap` before Alice.
4. Bob successfully swaps the 10 WETH from Alice to himself.

Recommendation: The limitation should be clearly stated to the user and warn them that they should only interact with other contracts that call the `Vaul.sol`. Alternatively, the `Vault.sol:_trasnferIn` function can call the `transferFrom` to collect the token directly instead of letting the upper contracts do the work.

QSP-19 Ignoring Return Value of ERC-20 Transfer

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Timelock.sol`, `GlpManaer.sol`, `RewardRouterV2.sol`

Description: In the `Timelock.sol:transferIn` function, the line `IERC20(_token).transferFrom(...)` ignores the return value of the generic ERC-20 transfer call. The implementation of the `transfer` and `transferFrom` functions of the ERC-20 tokens can potentially return `false` to indicate a failed transfer instead of reverting directly. Therefore, a failed transfer might sneak through the code without checking the return value.
We also identified the following instances with a similar pattern that are not vulnerable as they interact with tokens within the code base. Those tokens always revert on an error and do not return `false`:

1. `GlpManager._removeLiquidity()` (calling `IERC20(usdg).transfer(address(vault), usdgAmount);`)
2. `RewardRouterV2.acceptTransfer()` (calling `IERC20(esGmx).transferFrom(_sender, receiver, esGmxBalance);`)

Recommendation: Use `SafeTransferFrom` from the `SafeERC20` contract instead (see: OpenZeppelin doc).

QSP-20 Accessing Wrong Price Feed Data

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `VaultPriceFeed.sol`

Description: The `VaultPriceFeed.sol:getPrimaryPrice` function calls `priceFeed.getRoundData(rounded - i)` to get the price data for the previous round from the Chainlink oracles. However, from the official doc, the increase of the `rounded` might not be monotonic. So the `VaultPriceFeed.sol` logic can be wrong as it simply decreases 1 from the last round ID. Meanwhile, the doc also states: "To check the round, validate that the timestamp on that round is not 0". The current implementation checks against the price value `p` instead of the timestamp. (Note: the `VaultPriceFeed.sol` contract is out of the scope of this audit)

Recommendation: We recommend the following changes:

1. Use the `getpreviousroundid` function instead, see: doc.
2. Check the timestamp of the round to ensure the round is valid.

QSP-21 USDG Accounting Mismatch with Total Supply

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Vault.sol`

Description: The contract does the accounting for the USDG token with the mapping `usdgAmounts`. The mapping records the amount of USDG value for each token. The system will adjust the fee for `buyUSDG`, `sellUSDG`, and `swap` according to the recorded USDG amount and the targeted weight of each token with the function `getTargetUsdgAmount`.
However, the accounting can be inaccurate. During a `swap` transaction, the amount for the internal function `_decreaseUsdgAmount` can be insolvent without failing the transaction. In that case, the `usdgAmounts[_tokenOut]` will become zero, but `usdgAmounts[_tokenIn]` will increase the full `usdgAmount` within the following lines in the `swap` function. Now, the sum of `usdgAmounts` for all tokens will be more than the total supply of the USDG token.

```
_increaseUsdgAmount(_tokenIn, usdgAmount);
_decreaseUsdgAmount(_tokenOut, usdgAmount);
```

Exploit Scenario:

1. Given that `usdgAmounts[tokenA] = 10` and `usdgAmounts[tokenB] = 10`.
2. Alice swaps tokenB with tokenA for the equivalent of 15 USD.
3. The account will become `usdgAmounts[tokenA] = 25` and `usdgAmounts[tokenB] = 0`
4. The sum of the `usdgAmounts` mapping increased from 20 to 25 while the total supply for the USDG is still 20.

Recommendation: We recommend the team first clarify the business goal of the `usdgAmounts` accounting and determine the severity of the impact. The team should clarify what would happen if the `getFeeBasisPoints` function motivates wrongly and decides whether it needs fixing.

QSP-22 Risk of Inaccurate GLP Token Minting

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Vault.sol`, `GlpManager.sol`

Description: The `GlpManager.sol:getAum` function loops against all tokens in the `Vault.sol:allWhitelistedTokens` array. First, there is a concern of a DOS (denial of service) factor that if the governance adds enough tokens by `Vault.sol:setTokenConfig`, the loop can run out of gas. Secondly, the `Vault.sol:clearTokenConfig` function does not remove the token from the `allWhitelistedTokens` array. Thus, if the same token is cleared and set again, the token will duplicate in `allWhitelistedTokens`. The `GlpManager.sol:getAum` function will calculate the same token twice and add it to the final `aum` result. The problem will eventually impact the `GlpManager.sol:_addLiquidity` function and mint less GLP tokens (`aumInUsdg` will increase so that the `mintAmount` will decrease).

Exploit Scenario:

1. The governance adds a stablecoin, `USDStable`.
2. After a specific time, the `USDStable's` reputation goes down. The governance decides to remove the `USDStable` token from the pool.
3. However, the `USDStable` token re-gain its reputation and popularity after a while. The governance decides to add the token back.
4. The `GLP` will be minted with fewer tokens.

Recommendation: Ensure the token is removed from `allWhitelistedTokens` in the `Vault.sol:clearTokenConfig` function.

Update: According to the bug bounty (link), the team is aware of this and has excluded this issue from the rewards. Thus, we set the status as "Acknowledged".

```
Calling Vault.setTokenConfig, Vault.clearTokenConfig, Vault.setTokenConfig on the same token would lead to double counting of the token amounts in GlpManager, Vault.clearTokenConfig will not be used
```

QSP-23 AMM Price Excluded From Pricing Feed

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Vault.sol`

Description: In the `Vault.liqudationPosition` function, `includeAmmPrice` is set to `false` to prevent manipulated liquidation. At the end of the function, `includeAmmPrice` is set back to true. If a position exceeds the maximum leverage state, the position is automatically decreased. However, the `includeAmmprice` is not reset back to `true`. If a position is recently liquidated, all price data for all Vault's functions is obtained with `includeAmmPrice` set to `false`.

Recommendation: Set `includeAmmPrice` back to true before the early return in the condition `if (liquidationState == 2) {...}`.

Update: This issue has been fixed in the repo's `787d767` version. We recommend to deploy the fixed version of the contract.

QSP-24 Missing Position Data

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PositionManager.sol`

Description: In `PositionManager.executeDecreaseOrder` function, the function executes a decrease position order. After executing the decrease position order, `_sizeDelta` data is fetched from the `OrderBook`. However, in `OrderBook.executeDecreaseOrder`, the data stored in `decreaseOrders` gets deleted. Therefore, since decreasing order is executed before the position data is fetched, the `_sizeDelta` returned by `OrderBook.executeDecreaseOrder` is always zero. This impacts the data in the `DecreasePositionReferral` event as it can cause off-chain components to listen to the wrong value.

Recommendation: Change the order of the code in the `executeDecreaseOrder` function. Call `IOrderBook(orderBook).getDecreaseOrder` before `IOrderBook(orderBook).executeDecreaseOrder` to get the correct `_sizeDelta` data.

Update: This issue has been corrected in the repo's `787d767` version. We recommend to deploy the latest `PositionManager.sol` contract.

QSP-25 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Unresolved

File(s) affected: `GMX.sol`, `RewardTracker.sol`, `StakedGlp.sol`

Description: In `BaseToken.sol`, the `approve` function sets the allowance of a spender on behalf of the `msg.sender`. As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens. Following is the list of token contracts impacted: `GMX.sol`, `RewardTracker.sol`, and `StakedGlp.sol`.

Exploit Scenario:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) can be mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()` (see: OpenZeppelin ERC20). Furthermore, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value.

QSP-26 Missing Implementation of Vault Access Function in Timelock

Severity: *Informational*

Status: Unresolved

File(s) affected: `Timelock.sol`, `Vault.sol`

Description: Some of the `Vault` functions are only accessible by the `gov` address; the `gov` address is set to the `Timelock` contract. Since only `Timelock` contract can interact with the `Vault` functionalities, it is not possible to use the functions without implementation within the `Timelock`. It is unclear why certain implementations are missing out from the `Timelock` contract. Some of the functions introduce issues discussed in this report, which may be why the functions are missing in the `Timelock`. However, the `Timelock` contract can be replaced with another contract using the `Time.sol:signalSetGov` function.
The following functions are missing access implementation in the `Timelock.sol` contract:

- Missing access function for `Vault.sol:clearTokenConfig` function in `Timelock.sol`.
- Missing access function for `Vault.sol:setErrorController` function in `Timelock.sol`.
- Missing access function `Vault.sol:setInManagerMode` function in `Timelock.sol`.
- Missing access function `Vault.sol:setManager` function in `Timelock.sol`.
- Missing access function `Vault.sol:upgradeVault` function in `Timelock.sol`. If implemented, this function should have a time delay.
- Missing access function `GlpManager.sol:setAumAdjustment` function in `Timelock.sol`. If implemented, this function should have a time delay.

Recommendation: Update documentation to provide reasons why the missing functions are not implemented in the `Timelock` contract. If the functions are deprecated, inform the users that the function will not be used. If a function is used in certain situations, inform the users when and in what situation, the function will be used.

QSP-27 Functions Available While Uninitialized

Severity: *Informational*

Status: Unresolved

File(s) affected: `Vault.sol`, `OrderBook.sol`, `RewardRouterV2.sol`, `RewardTracker.sol`

Description: The `initialize` function is required before the contract can properly function; the initialization requirement is identified in multiple contracts in the codebase. However, functions are accessible even if `initialize` is not called. The following is the list of impacted contracts: `Vault.sol`, `OrderBook.sol`, `RewardRouterV2.sol`, `RewardTracker.sol`.

Recommendation: Ensure that other functions can only be called after initialization.

QSP-28 Missing Upgrade Path for Vault Implementation

Severity: *Informational*

Status: Unresolved

File(s) affected: `All Contracts`

Description: The current `Timelock` contract is missing access to the function `Vault.upgradeVault`. With the current `Timelock` and `Vault` contracts, it is not possible to upgrade the `Vault` contract. This might explain the differences identified between the repo and the deployed contracts.
In order to upgrade the `Vault` contract, the team needs to deploy a new `Timelock` contract with access to the `Vault.upgradeVault` function to migrate the assets to the new `Vault`. When migrating assets between the vaults, the team needs to disable access to GLP tokens because the value of the GLP is dynamically determined by the Vault's value in `GlpManager.getAum`.
New contracts such as `GlpManager`, `OrderBook`, `PositionManager`, `PositionRouter`, and `Router` need to be deployed because the `vault` address is only set in the `constructor` or in the `initialize` function.
If a future issue is identified in the `Vault` contract, the protocol may be down for a while since the upgrade path is unclear.

Recommendation: Document the operational plan for contract upgrades. Also, consider using the UUPS proxy pattern to upgrade Vault's implementation.

QSP-29 Critical Role Transfers Not Following Two-Step Pattern

Severity: *Informational*

Status: Unresolved

File(s) affected: `Vault.sol`, `BaseToken.sol`, `RewardTracker.sol`, `BasePositionManager.sol`, `OrderBook.sol`, `GlpManager.sol`, `Router.sol`

Description: In multiple contracts, roles can be transferred to another address simply by calling the corresponding `*.set*()` function. These functions immediately transfer a high-level privilege to a new address in a single transaction, which can be risky from a security perspective, as **providing a faulty address may lockout that role from future calls, rendering the contract potentially useless.**

1. `Vault.setGov()`
2. `BaseToken.setGov()`
3. `RewardTracker.setGov()`
4. `BasePositionManager.setGov()`
5. `OrderBook.setGov()`
6. `GlpManager.setGov()`
7. `Router.setGov()`

A more secure pattern for such privilege transfers requires the new pending addresses to issue an `acceptAdmin()` function call before finalizing the transfer. Note that this pattern is common even with the use of timelocks, such as the Compound Timelock contract (see functions `setPendingAdmin()` and `acceptAdmin()`).

Recommendation: Require the pending new role address to make an `acceptAdmin()` function call before fully transferring the privilege.

QSP-30 Unlocked Pragma

Severity: *Informational*

Status: Unresolved

File(s) affected: `BasePositionManager.sol`, `OrderBook.sol`, `PositionManager.sol`, `PositionRouter.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

QSP-31 Reentrancy Risk

Severity: *Informational*

Status: Unresolved

File(s) affected: `Vault.sol`, `OrderBook.sol`, `GlpManager.sol`, `Router.sol`

Related Issue(s): SWC-107

Description: A reentrancy vulnerability is a scenario where an attacker can repeatedly call a function from itself, unexpectedly leading to potentially disastrous results. Reentrancy can potentially occur on the token transfer if non-standard ERC20 is integrated (e.g., ERC777).
The following is the place(s) that we identified that misses reentrancy protection:

1. In the `Vault.withdrawFees` function, it calls `Vault._transferOut`. The `_transferOut` function updates the `tokenBalances[_token]` storage after the token transfer.

Also, we noticed that several places are protected by the `nonReentrant` modifier but not following the checks-effects-interactions pattern. Future code changes and audits should keep in mind to not re-introduce risk in the following places:

1. `OrderBook.createSwapOrder()`: Call to external contracts (`IERC20(_token).safeTransferFrom()`), before modifying `swapOrdersIndex[]` and `swapOrders[]`. However, the function is protected by `nonReentrant`.
2. `OrderBook.createIncreaseOrder()`: Call to external contracts (`IERC20(_token).safeTransferFrom()`), before modifying `increaseOrdersIndex[]` and `increaseOrders[]`. However, the function is protected by `nonReentrant`.
3. `OrderBook.executeIncreaseOrder()`: Call to external contracts (`IERC20(order.purchaseToken).safeTransfer()` and `IERC20(order.collateralToken).safeTransfer()`), before modifying state variables in `Vault.sol` (`positions[]`). However, the function is protected by `nonReentrant`.
4. `Vault._transferIn()`: Call to external contracts (`IERC20(_token).balanceOf`)), before modifying state variables (`tokenBalances[]`). However, the function is private, and all its callers are protected by `nonReentrant`.
5. `GlpManager._addLiquidity()`: Call to external contracts (`IERC20(_token).safeTransferFrom()`), before modifying state variables (`lastAddedAt[]`). However, the function is private, and all its callers are protected by `nonReentrant`.
6. `Router.directPoolDeposit()`: Call to external contracts (`IERC20(_token).safeTransferFrom()`), before modifying state variables in `Vault.sol`. However, called function `IVault(vault).directPoolDeposit()` is protected by `nonReentrant`.

Note that we did not find any obvious exploit from this.

Recommendation: Add the `nonReentrant` modifier to protect the function from reentrancy or enforce the code to follow the checks-effects-interactions pattern.

QSP-32 Concerning Unexecuted Time-Locked Actions

Severity: *Informational*

Status: Unresolved

File(s) affected: `Timelock.sol`

Description: The `Timelock.sol` contract involves two-step executions. The admin must first "signal" the action and execute it after the `buffer` period. However, if action is "signaled" but never executed, it can cause unexpected risks to the users.

Recommendation: We recommend adding a deadline to the signaled actions. So once the deadline passes, the admin can no longer execute the action.

QSP-33 Risk of Self Collateralization

Severity: *Informational*

Status: Unresolved

File(s) affected: `Vault.sol`, `Timelock.sol`

Description: The `Vault.sol : setTokenConfig` function adds a token to the whitelisted tokens. However, it does not validate if `USDG` or `GLP` are added to the whitelisted token or not. Both `USDG` and `GLP` are backed by the tokens in the pool. If those tokens are whitelisted, then a user can buy `USDG` with `USDG`, causing the printing of new tokens based on their value and breaking the tokenomics.

Recommendation: Add validation in the `Vault.sol : setTokenConfig` function to prevent the addition of the `USDG` and `GLP` tokens to the whitelist.

QSP-34 Swap Pricing Disabled

Severity: *Informational*

Status: Fixed

File(s) affected: `Vault.sol`

Description: In `Vault.buyUSDG` function, the function does not enable swap pricing by setting `useSwapPricing` to `true`. This parameter is passed to `VaultPriceFeed.getPrice` function. Based on the deployed version of `VaultPriceFeed`, this parameter is not used in the `VaultPriceFeed.getPrice` function. Therefore, no current impact is identified by this issue.

Recommendation: Set the `useSwapPrice` as `true` at the beginning of the `Vault.buyUSDG` function.

Update: This issue has been corrected in the repo's `787d767` version. We recommend to deploy the fixed version of the contract.

QSP-35 Privileged Roles and Ownership

Severity: Informational

Status: Unresolved

File(s) affected: `Vault.sol`, `BaseToken.sol`, `MintableBaseToken.sol`, `RewardTracker.sol`, `BasePositionManager.sol`, `PositionManager.sol`, `OrderBook.sol`, `PositionRouter.sol`, `GlpManager.sol`, `Router.sol`, `RewardRouterV2.sol`

Description: Specific contracts have state variables, e.g., `owner`, which provide specific addresses with privileged roles. Such roles may pose a risk to end-users. Note that for those with higher severity, we have also listed in another issue, "Ability to Rug-Pull".
The `Vault.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following listed functions!**)
  - Initialize important contract addresses and variables (`router`, `priceFeed`, `fundingRateFactor`, ...) via calling `initialize()`.
  - Designate an address with the `errorController` role (Who may modify existing require error messages/add new ones) by calling `setErrorController()`.
  - Set/Unset `inManagerMode`, in which case only certain addresses may buy/sell "USDG", by calling `setInManagerMode()`.
  - Add/Remove addresses from the `isManager[]` role by calling `setManager()`.
  - Set/Unset `inPrivateLiquidationMode`, in which case only certain addresses may liquidate positions, by calling `setInPrivateLiquidationMode()`.
  - Add/Remove addresses from the `isLiquidator[]` role by calling `setLiquidator()`.
  - Globally enable/disable swaps for everyone (currently enabled) by calling `setIsSwapEnabled()`.
  - Globally enable/disable leverage for everyone (currently false) by calling `setIsLeverageEnabled()`.
  - Set a maximally allowed gas price for increasing/decreasing positions (currently `0`, gas price is not checked) by calling `setMaxGasPrice()`.
  - Change the vault price feed address by calling `setPriceFeed()`.
  - Change the maximally allowed leverage (currently `1000000`, given `BASIS_POINTS_DIVISOR = 10000`, this leverage equals 1x) by calling `setMaxLeverage()`.
  - Set arbitrary buffer amounts for tokens, preventing performing swaps beyond a certain threshold by calling `setBufferAmount()`.
  - Change different protocol fees up to their corresponding maxima by calling `setFees()`.
  - Change funding rate related variables (interval, factor, stable coin factor) by calling `setFundingRate()`.
  - Change/Clear token-related configurations (whitelisted tokens, token decimals, ...) by calling `setTokenConfig()`/`clearTokenConfig()`.
  - Withdraw protocol fees to an arbitrary address by calling `withdrawFees()`.
  - Manually increase/decrease the "USDG" amount per token by calling `setUsdgAmount()`.
  - **Transfer an arbitrary amount of any token from the Vault to an arbitrary address by calling** `upgradeVault()`.
- `isManager[]`, as set through `setManager()` by `gov`:
  - Be the only one(s) able to buy "USDG", when in `inManagerMode` mode (currently the case), by calling `buyUSDG()`.
  - Be the only one(s) able to sell "USDG", when in `inManagerMode` mode (currently the case), by calling `sellUSDG()`.
- `isLiquidator[]`, as set through `setLiquidator()` by `gov`:
  - Be the only one(s) able to liquidate positions, when in `inPrivateLiquidationMode` mode (currently the case), by calling `liquidatePosition()`.

The `BaseToken.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following listed functions!**).
  - **Change the token name and symbol string at any time** by calling `setInfo()`.
  - Change the yield tracker address by calling `setYieldTrackers()`.
  - Add/Remove arbitrary addresses from the `admins[]` role by calling `addAdmin()`/`removeAdmin()`.
  - Withdraw arbitrary amounts of any token from the contract to an arbitrary address by calling `withdrawToken()`.
  - Set/Unset private transfer mode () by calling `setInPrivateTransferMode()`.
  - Allow arbitrary addresses to be "handlers", which in turn can arbitrarily move tokens between accounts by calling `setHandler()`.
- `admins[]`, as initialized during the `constructor()` execution:
  - Add/Remove accounts from the non-staking accounts pool/list by calling `addNonStakingAccount()`/`removeNonStakingAccount()`.
  - **Recover any pending claims for any account to an arbitrary address** by calling `recoverClaim()`.

The `MintableBaseToken.sol` contract contains the following privileged roles:

- `isMinter[]`, as set through `setMinter()` by `gov`:
  - Mint arbitrarily many tokens to any address by calling `mint()`.
  - Burn arbitrarily many tokens from any address by calling `burn()`.

The `RewardTracker.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following functions!**).
  - Initialize the list of allowed deposit tokens and distributor addresses by calling `initialize()`.
  - Add/Remove tokens from the whitelisted `isDepositToken` array by calling `setDepositToken()`.
  - Set/Unset private transfer/staking/claiming mode by calling `setInPrivateTransferMode()`, `setInPrivateStakingMode()` and `setInPrivateClaimingMode()`, respectively.
  - Allow arbitrary addresses to be "handlers", which in turn can arbitrarily move tokens between accounts by calling `setHandler()`.
  - Withdraw arbitrary amounts of any token from the contract to an arbitrary address by calling `withdrawToken()` (in combination with `setDepositToken()` if the token is a deposit token, by temporarily making it not a deposit token).

The `BasePositionManager.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following functions!**).
  - Assign a new `admin` address by calling `setAdmin()`.
  - Call the `approve()` function for any given token address by calling `approve()`.
  - Transfer an arbitrary amount of ether from the contract to another address by calling `sendValue()`.
- `admin`, as initialized during the `constructor()` execution to `msg.sender`:
  - Set an arbitrary deposit fee (**0-100%**!) by calling `setDepositFee()`.
  - Set an arbitrary position increase buffer (**down to 0 BPS**!) by calling `setIncreasePositionBufferBps()`.
  - Change the referral storage contract address by calling `setReferralStorage()`.
  - Arbitrarily change the maximum size of long/short positions per token (**down to 0**) by calling `setMaxGlobalSizes()`.
  - Withdrawing any accumulated amount in `feeReserves[]` for a given token to an arbitrary address by calling `withdrawFees()`.

The `PositionManager.sol` contract contains the following privileged roles:

- `admin`, as initialized during the `constructor()` execution to `msg.sender`:
  - All of the privileged roles mentioned above for `BasePositionManager.admin`, as it inherits from it.
  - Add arbitrary addresses to have the `isOrderKeeper[]` role by calling `setOrderKeeper()`.
  - Add arbitrary addresses to have the `isLiquidator[]` role by calling `setLiquidator()`.
  - Add arbitrary addresses to have the `isPartner[]` role by calling `setPartner()`.
  - Activate/Deactivate legacy mode (disabling/enabling access controls for increasing/decreasing positions functions) by calling `setInLegacyMode()`.
  - Control whether or not long order increasing functions should be validated (to not be decreasing) by calling `setShouldValidateIncreaseOrder()`.
- `isOrderKeeper[]`, as set through `setOrderKeeper()` by `admin`:
  - Execute order book swaps by calling `executeSwapOrder()`.
  - Execute order book order increases by calling `executeIncreaseOrder()`.
  - Execute order book order decreases by calling `executeDecreaseOrder()`.
- `isLiquidator[]`, as set through `setLiquidator()` by `admin`:
  - Liquidate arbitrary positions by calling `liquidatePosition()`.
- `isPartner[]`, as set through `setPartner()` by `admin`:
  - Increase a position using only ERC20 tokens (and potentially perform a swap) by calling `increasePosition()`.
  - Increase a position using ETH (and potentially perform a swap) by calling `increasePositionETH()`.
  - Decrease a position using only ERC20 tokens by calling `decreasePosition()`.
  - Decrease a position using ETH by calling `decreasePositionETH()`.
  - Decrease a position using only ERC20 tokens and perform a swap by calling `decreasePositionAndSwap()`.
  - Decrease a position using ETH and perform a swap by calling `decreasePositionAndSwapETH()`.
- `inLegacyMode`, as set through `setInLegacyMode()` by `admin`:
  - When set, all the same functions as by the role `isPartner[]` above can be called **by anyone**.

The `OrderBook.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following functions!**).
  - Initialize state variables (`router`, `vault`, ..) by calling `initialize()`.
  - Set **arbitrary high/low** minimum order book ETH execution fees by calling `setMinExecutionFee()`.
  - Set an **arbitrary high/low** minimum order increase sizes by calling `setMinPurchaseTokenAmountUsd()`.

The `PositionRouter.sol` contract contains the following privileged roles:

- Same privileges and roles as `BasePositionManager.sol` listed above, as it inherits from it.
- In addition, `admin` has the following privileges:
  - Add/Remove arbitrary addresses from the `isPositionKeeper[]` role by calling `setPositionKeeper()`.

-
  - Set **arbitrary high/low** minimum position ETH creation fees by calling `setMinExecutionFee()`.
  - Enable/Disable leverage by calling `setIsLeverageEnabled()`.
  - Set minimum/maximum position (block) delay execution times by calling `setDelayValues()`.
  - Set increase/decrease position related internal bookkeeping indices by calling `setRequestKeysStartValues()`.
- `isPositionKeeper[]`, as set through `setPositionKeeper()` by `admin`:
  - Execute the creation of position increases by calling `executeIncreasePositions()`.
  - Execute the creation of position decreases by calling `executeDecreasePositions()`.

The `GlpManager.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following functions!**).
  - Enable/Disable private mode (restricting adding/removing liquidity) by calling `setInPrivateMode()`.
  - Add/Remove arbitrary addresses to the `isHandler[]` role by calling `setHandler()`.
  - Set the cooldown duration up to `MAX_COOLDOWN_DURATION` (48 hours) for removing liquidity by calling `setCooldownDuration()`.
  - Set AUM (Asset Under Management) computation corrections (addition/deduction deltas) by calling `setAumAdjustment()`.
- `isHandler[]`, as set through `setHandler()` by `gov`:
  - Add/Remove liquidity **for arbitrary other accounts** by calling `addLiquidityForAccount()` and `removeLiquidityForAccount()`, respectively.

The `Router.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following functions!**).
  - Add/Remove addresses from the `plugin[]` role by calling `addPlugin()`/`removePlugin()`.
- `plugin[]`, as set through `addPlugin()` by `gov`:
  - Call `safeTransferFrom()` for any token address and arbitrary parameters by calling `pluginTransfer()`.
  - Increase/Decrease a position on ones behalf in the `Vault.sol` contract by calling `pluginIncreasePosition()`/`pluginDecreasePosition()`.

The `RewardRouterV2.sol` contract contains the following privileged roles:

- `gov`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign a new `gov` address by calling `setGov()` (Or renouncing the role by setting it to an uncontrolled address, like `address(0)` **and thereby preventing any future calls to the following functions!**).
  - Initialize important contract addresses (`gmx`, `stakedGmxTracker`, `glpManager`, ...) by calling `initialize()`.
  - Withdraw arbitrary tokens from the contract by calling `withdrawToken()`.
  - Stake multiple amounts of "GMX" on behalf of other accounts by calling `batchStakeGmxForAccount()`.
  - Stake "GMX" on behalf of another account by calling `stakeGmxForAccount()`.
  - Compound (claim and re-stake) "GMX" and "GLP" on behalf of another account by calling `compoundForAccount()`.
  - Compound (claim and re-stake) multiple amounts of "GMX" and "GLP" on behalf of other accounts by calling `batchCompoundForAccounts()`.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

QSP-36 Stop Users From Redeeming GLP

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `GLP.sol`, `Timelock.sol`, `BaseToken.sol`

Description: GLP's base implementation inherits from `BaseToken.sol` which is a custom ERC20 implementation. Due to the custom implementation, the admin can prevent the user from transferring GLP tokens using the function `setInPrivateTransferMode` in `BaseToken.sol`.

1. The function to enable `inPrivateTransferMode` is only accessible by the `gov`; the `gov` address is set to the `Timelock` contract. This `setInPrivateTransferMode(...)` function is accessible by the admin and no time delay is needed to enable this mode. If this mode is enabled, the team can stop users from redeeming (transferring) GLP tokens. During `inPrivateTransferMode` mode, only handlers can access the users' funds. Only admins can enable transfer again.
2. Allowing handlers to access user funds also introduces a single point of failure, and all funds can be drained in the case of a compromised handler account.

Recommendation:

1. Remove `BaseToken.setInPrivateTransferMode` or evaluate the need for `inPrivateTransferMode` mode and update the documentation to inform the users.
2. Remove the handler's access to users' funds or get approval from users before enabling handler access.

QSP-37 `BaseToken.sol` Allows Changing `_name` and `_symbol` After Deployment

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `BaseToken.sol`

Description: Contract `BaseToken.sol` (and thereby tokens inheriting from it, like `GMX`), allows for retro-actively changing the token `_name` and `_symbol` state variables, by calling `setInfo()`, given the caller is `gov`.
This may lead to unexpected behavior with other interacting contracts/(DeFi) platforms.

Recommendation: Clarify if this is intended behaviour and consider removing this functionality, as it is also nowhere used.

QSP-38 Unclear Intentions and Specs

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `OrderBook.sol`, `Vault.sol`

Description: We noticed some places of the implementation could be confusing, and the team should clarify the spec:

1. In the contract `OrderBook.sol`, the function `cancelSwapOrder` will always unwrap if `order.path[0] == weth`. However, a swap order can be created with WETH instead of ETH when calling the `createSwapOrder` function. In that case, the return of the token during the cancellation will still be unwrapped. The team should clarify whether this is intended. If so, this should be documented to the users. Otherwise, consider rejecting `createSwapOrder` with WETH as the `path[0]` input.
2. The technical overview doc states that the `PositionRouter.sol` contract helps in reducing the front-running issues with a two-step process. During the process, a keeper requests the index price from an aggregate of exchanges and then executes the position at the current index price. However, the implementation of `PositionRouter.sol:executeIncreasePositions` and `PositionRouter.sol:executeDecreasePositions` do not allow the keeper to pass in the aggregated index price. This disobeys the statement of the documentation.
3. The usage of the functions `Vault.sol:getRedemptionCollateral` and `Vault.sol:getRedemptionCollateralUsd` is unclear. None of the contracts call the functions. The team should clarify the intention of the functions.

Recommendation: Follow the recommendation stated in the description and clarify the spec.

Automated Analyses

Slither

Slither analyzed all of the contracts in the repository of the latest `master` branch (commit: `787d767e033c4`) and found 2429 results. Most of them are out-of-scope or false-positive. We have added the valid ones to the report.

Adherence to Specification

1. Not all on-chain contracts are listed in the documentation (i.e. compare against the contracts listed in the corresponding bug bounty under "Assets in scope").

Code Documentation

1. Every function should at least have a short description of its purpose, its input parameters and its return value(s) if any. Functions in the code base do not have such comments which increase the effort of maintainability and the probability of human error when subsequent features are added/modified/removed.
2. For all contracts with privileged roles such as `gov`, document the expected contract or setup for the privileged role.
3. Add NatSpec (see: doc) code document for all public and external functions.
4. Consider adding a warning on using the `Vault.sol_decreaseUsdgAmount` function. The function will decrease **until zero** if the `_amount` input is larger than the value from `usdgAmounts[_token]`.
5. Explain the formula used to calculate the `GlpManager.sol:getAum` function.
6. Consider adding an explanation of each field of the critical structs for the `OrderBook.sol`, including the following structs: `IncreaseOrder`, `DecreaseOrder`, and `SwapOrder`.
7. Add documentation for the `IYieldTracker`. The functions should state the expected behaviors, such as the token flow.
8. Explain the background and the reasoning of the "legacy" mode in the `PositionManager.sol` contract.
9. Document the limit for the `RewardTracker.sol` contract, especially on the "deposit token" (or the staking token). The contract does not support deflationary tokens or rebase-able tokens. Also, all the deposit tokens must share the same decimal, and the value per token should be the same. The `RewardTracker.sol` contract would sum the amount from all staked tokens naively.

Adherence to Best Practices

1. The logic used to calculate average short prices in the `Vault` is duplicated in multiple functions such as `getGlobalShortDelta`, `getNextAveragePrice`, `getNextGlobalShortAveragePrice`, and `getgetDelta`. Consider refactoring the duplicated logic in a single function to be used.
2. Before transferring funds, add additional verification to confirm the recipient is not a zero address to prevent loss of funds.
3. Consider restricting the accessibility of the `Vault.increasePosition` and `Vault.decreasePosition` functions if the functions are only accessible by certain contracts. With the current implementation, it is possible to decrease position without using `PositionManager` and `PositionRouter`.
4. Consider renaming the `Reader.setConfig` function to `Reader.setMaxGlobalShortSizes` to reflect the function's objective.

5. To facilitate logging, it is recommended to index address parameters within events by prepending the `indexed` keyword. Multiple contracts are impacted (`Vault.sol`, `RewardRouterV2.sol`, `PositionManager.sol`, ...).
6. Contract `StakedGlp.sol` unnecessarily imports `IRewardTracker.sol` twice. For improved readability and code quality, it is advised to remove duplicate or unused code.
7. Merge the functionality of the `initialize` function into `constructor` if the contract is not an upgradeable one. Also, with the simplification, the implementation can remove the `isInitialized` storage variable. The following is the list of contracts that can remove the `initialize` function: `Vault.sol`, `OrderBook.sol`, `RewardTracker.sol`, `RewardRouterV2.sol`.
8. Change the default value for the `Vault.sol:isLeverageEnabled` to false in the variable declaration. The flag is used so only the `PositionManger.sol` can call the function.
9. Consider replacing the function `Vault.sol:_onlyGov` with a modifier `onlyGov` instead. Similar applies to `Vault.sol:_validateManager` (-> `onlyManager`). It is common to have basic authorization checks in the modifier.
10. Replace magic numbers for the error codes with predefined constants or Enum in `Vault.sol`. The magic number ranges from 1 to 55 over the file.
11. Simplify the `Orderbook.sol:validatePositionOrderPrice` function to not return the `isPriceValid` boolean flag and remove the `_raise` boolean input. All the code calls the function with the `_raise` as true and does not use the returned `isPriceValid` flag.
12. Replace magic numbers with constants in `OrderBookReader.sol:getIncreaseOrders`. The number `5` and `3` in the line `Vars memory vars = Vars(0, 0, _account, 5, 3);`.
13. Consider passing in the actual governance contract address directly in the `constructor` of the `Router` and the `Vault` contract. This can reduce the operation step and potentially remove the need for the `setGov` function.
14. In `Reader.sol`, consider extracting the logic regarding `feeBasisPoints` and reuse between the `getFeeBasisPoints` and `getAmountOut` functions.
15. Remove the unnecessary storage variable `Vault.sol:useSwapPricing`. The value of the variable is passed to the `IVaultPriceFeed(priceFeed).getPrice(...)` function in `Vaul.sol:getMaxPrice` and `Vault.sol:getMinPrice`. First, it does not need to be a storage variable. The function calling it can pass a hardcoded value on each function instead (e.g., `Vault.sol:swap`, `Vault.sol:sellUSDG`, and `Vault.sol:buyUSDG`) to save gas. Note that for the deployed contracts, the `Vault.sol:buyUSDG` function, unlike other functions, forgets to set the value as `true` (Note: This is later fixed, and the latest master commit adds the line to set the value). Aside from that, the `VaultPriceFeed.sol:getPrice` will not use that input. So it is a useless variable.
16. In the `Reader.sol:getMaxAmountIn` function, the line `amountIn = availableAmount.mul(priceOut).div(priceIn).mul(10 ** tokenInDecimals).div(10 ** tokenOutDecimals)` does division (`.div(priceIn)`) before multiplication (`.mul(10 ** tokenInDecimals)`). We recommend to change the order for a more precise result: `amountIn = availableAmount.mul(priceOut).mul(10 ** tokenInDecimals).div(priceIn).div(10 ** tokenOutDecimals)`.

Test Results

Test Suite Results

See the setup of running the coverage.

```
  TokenManager
     ✓ inits (104ms)
     ✓ signalApprove (59ms)
     ✓ signApprove (218ms)
     ✓ approve (878ms)
     ✓ signalApproveNFT (66ms)
     ✓ signApproveNFT (169ms)
     ✓ approveNFT (541ms)
     ✓ signalApproveNFTs (43ms)
     ✓ signApproveNFTs (178ms)
     ✓ approveNFTs (531ms)
     ✓ receiveNFTs (379ms)
     ✓ signalSetAdmin (68ms)
     ✓ signSetAdmin (194ms)
     ✓ setAdmin (230ms)
     ✓ signalSetGov (49ms)
     ✓ signSetGov (185ms)
     ✓ setGov (469ms)

  GlpManager
     ✓ inits
     ✓ setGov (76ms)
     ✓ setHandler (70ms)
     ✓ setCooldownDuration (81ms)
     ✓ setAumAdjustment (361ms)
addLiquidity gas used 692513
removeLiquidity gas used 598519
     ✓ addLiquidity, removeLiquidity (6120ms)
     ✓ addLiquidityForAccount, removeLiquidityForAccount (2578ms)

  OrderBook, cancelMultiple
     ✓ cancelMultiple (480ms)

  OrderBook, decrease position orders
     ✓ Create decrase order, bad fee (40ms)
createDecraseOrder gas used 298843
     ✓ Create decrease order, long (46ms)
updateDecreaseOrder gas used 53771
     ✓ updateDecreaseOrder (156ms)
createDecreaseOrder gas used 278931
     ✓ Create decrease order, short (60ms)
     ✓ Create two orders (77ms)
     ✓ Execute decrease order, invalid price (851ms)
     ✓ Execute decrease order, non-existent (38ms)
executeDecreaseOrder gas used 380603
     ✓ Execute decrease order, long (553ms)
executeDecreaseOrder gas used 386840
     ✓ Execute decrease order, short, BTC (536ms)
createSwapOrder 278907
executeDecreaseOrder gas used 435078
     ✓ Execute decrease order, long, BNB (631ms)
cancelDecreaseOrder gas used 90367
     ✓ Cancel decrease order (150ms)

  OrderBook
     ✓ setGov (75ms)
     ✓ set* (52ms)
     ✓ initialize, already initialized

  OrderBook, increase position orders
     ✓ createIncreaseOrder, bad input (326ms)
     ✓ createIncreaseOrder, two orders (176ms)
createIncreaseOrder gas used 416336
     ✓ createIncreaseOrder, pay WETH (90ms)
createIncreaseOrder gas used 385003
     ✓ createIncreaseOrder, pay BNB (73ms)
createIncreaseOrder gas used 440688
     ✓ createIncreaseOrder, long A, transfer and purchase A (95ms)
createIncreaseOrder gas used 738856
     ✓ createIncreaseOrder, long A, transfer A, purchase B (239ms)
createIncreaseOrder gas used 420812
     ✓ createIncreaseOrder, short A, transfer B, purchase B (94ms)
createIncreaseOrder gas used 718980
```

```
     ✓ createIncreaseOrder, short A, transfer A, purchase B (230ms)
updateIncreaseOrder gas used 48035
     ✓ updateIncreaseOrder (135ms)
cancelIncreaseOrder gas used 111198
     ✓ cancelOrder (187ms)
cancelIncreaseOrder gas used 95207
     ✓ cancelOrder, pay BNB (162ms)
     ✓ executeOrder, non-existent order
     ✓ executeOrder, current price is invalid (1912ms)
executeIncreaseOrder gas used 576094
     ✓ executeOrder, long, purchase token same as collateral (342ms)
     ✓ executOrder, 2 orders with the same position (648ms)
executeIncreaseOrder gas used 796842
     ✓ executeOrder, long, swap purchase token to collateral (466ms)
executeIncreaseOrder gas used 592782
     ✓ executeOrder, short, purchase token same as collateral (314ms)
executeIncreaseOrder gas used 796133
     ✓ executeOrder, short, swap purchase token to collateral (441ms)
executeIncreaseOrder gas used 793633
     ✓ executeOrder, short, pay BNB, no swap (616ms)
     ✓ createIncreaseOrder, bad path

  OrderBook, swap orders
     ✓ createSwapOrder, bad input (227ms)
createSwapOrder 358079
     ✓ createSwapOrder, DAI -> BTC (85ms)
createSwapOrder 333703
     ✓ createSwapOrder, WBNB -> DAI (107ms)
createSwapOrder 302370
     ✓ createSwapOrder, BNB -> DAI (122ms)
createSwapOrder 338191
     ✓ createSwapOrder, DAI -> WBNB, shouldUnwrap = false (118ms)
createSwapOrder 358091
createSwapOrder 306791
     ✓ createSwapOrder, two orders (148ms)
canceSwapOrder 108449
     ✓ cancelSwapOrder, tokenA != BNB (213ms)
canceSwapOrder 92419
     ✓ cancelSwapOrder, tokenA == BNB (84ms)
updateSwapOrder 55838
     ✓ updateSwapOrder (142ms)
executeSwapOrder 429568
     ✓ executeSwapOrder, triggerAboveThreshold == false (394ms)
executeSwapOrder 379241
     ✓ executeSwapOrder, triggerAboveThreshold == false, DAI -> WBNB, shouldUnwrap = false (233ms)
executeSwapOrder 445319
     ✓ executeSwapOrder, triggerAboveThreshold == true (381ms)
executeSwapOrder 625549
     ✓ executeSwapOrder, triggerAboveThreshold == true, BNB -> DAI -> BTC (500ms)
executeSwapOrder 426489
     ✓ executeSwapOrder, triggerAboveThreshold == true, USDG -> BTC (663ms)
executeSwapOrder 605941
     ✓ executeSwapOrder, triggerAboveThreshold == true, USDG -> DAI -> BTC (891ms)
executeSwapOrder 652149
     ✓ executeSwapOrder, triggerAboveThreshold == true, USDG -> BNB -> BTC (1685ms)
executeSwapOrder 368597
     ✓ executeSwapOrder, triggerAboveThreshold == true, BTC -> USDG (443ms)
     ✓ complex scenario (905ms)

  PositionManager
     ✓ inits (43ms)
     ✓ setDepositFee (71ms)
     ✓ approve (51ms)
     ✓ setOrderKeeper (57ms)
     ✓ setLiquidator (62ms)
     ✓ setPartner (56ms)
     ✓ setInLegacyMode (60ms)
     ✓ setShouldValidateIncreaseOrder (80ms)
     ✓ increasePosition and decreasePosition (3705ms)
     ✓ increasePositionETH and decreasePositionETH (2516ms)
     ✓ increasePositionETH with swap (1739ms)
     ✓ increasePosition and increasePositionETH to short (1089ms)
     ✓ decreasePositionAndSwap and decreasePositionAndSwapETH (3424ms)
     ✓ deposit collateral for shorts (870ms)
     ✓ executeSwapOrder (352ms)
     ✓ executeIncreaseOrder (3968ms)
     ✓ executeDecreaseOrder (925ms)
     ✓ liquidatePosition (1484ms)

  PositionRouter
     ✓ inits (57ms)
     ✓ setAdmin (57ms)
     ✓ setDepositFee (82ms)
     ✓ setIncreasePositionBufferBps (71ms)
     ✓ setReferralStorage (75ms)
     ✓ setMaxGlobalSizes (159ms)
     ✓ withdrawFees (1520ms)
     ✓ approve (56ms)
     ✓ sendValue (57ms)
     ✓ setPositionKeeper (109ms)
     ✓ setMinExecutionFee (61ms)
     ✓ setIsLeverageEnabled (68ms)
     ✓ setDelayValues (88ms)
     ✓ setRequestKeysStartValues (68ms)
     ✓ increasePosition acceptablePrice long (548ms)
     ✓ increasePosition minOut long (475ms)
     ✓ validateExecution (1083ms)
     ✓ validateCancellation (752ms)
     ✓ maxGlobalLongSize (1051ms)
     ✓ decreasePosition acceptablePrice long (965ms)
     ✓ decreasePosition minOut long (1342ms)
     ✓ increasePosition acceptablePrice short (538ms)
     ✓ maxGlobalShortSize (1079ms)
     ✓ decreasePosition acceptablePrice short (825ms)
createIncreasePosition gas used 518024
executeIncreasePosition gas used 966153
cancelIncreasePosition gas used 149588
createIncreasePosition gas used 458059
executeIncreasePosition gas used 673062
     ✓ createIncreasePosition, executeIncreasePosition, cancelIncreasePosition (3005ms)
createIncreasePositionETH gas used 471251
executeIncreasePosition gas used 946413
cancelIncreasePosition gas used 137494
createIncreasePosition gas used 451109
executeIncreasePosition gas used 670905
     ✓ createIncreasePositionETH, executeIncreasePosition, cancelIncreasePosition (2833ms)
createIncreasePosition gas used 518024
executeIncreasePosition gas used 966153
createDecreasePosition gas used 396770
executeDecreasePosition gas used 559208
executeDecreasePosition gas used 764483
     ✓ createIncreasePosition, createDecreasePosition, executeDecreasePosition, cancelDecreasePosition (4648ms)
     ✓ executeIncreasePositions, executeDecreasePositions (11084ms)

  Router
     ✓ setGov (57ms)
     ✓ addPlugin (73ms)
     ✓ removePlugin (112ms)
     ✓ approvePlugin (45ms)
     ✓ denyPlugin (94ms)
     ✓ pluginTransfer (189ms)
     ✓ pluginIncreasePosition (207ms)
     ✓ pluginDecreasePosition (119ms)
buyUSDG gas used 397836
     ✓ swap, buy USDG (298ms)
sellUSDG gas used 397848
     ✓ swap, sell USDG (627ms)
swap gas used 412448
     ✓ swap, path.length == 2 (608ms)
swap gas used 501735
     ✓ swap, path.length == 3 (1560ms)
increasePosition gas used 879398
     ✓ swap, increasePosition (1360ms)
```

```
    ✓ decreasePositionAndSwap (1464ms)
    ✓ decreasePositionAndSwapETH (1271ms)

  Vault.averagePrice
    ✓ position.averagePrice, buyPrice != markPrice (4290ms)
    ✓ position.averagePrice, buyPrice == markPrice (2327ms)
    ✓ position.averagePrice, buyPrice < averagePrice (1464ms)
    ✓ long position.averagePrice, buyPrice == averagePrice (962ms)
    ✓ long position.averagePrice, buyPrice > averagePrice (974ms)
    ✓ long position.averagePrice, buyPrice < averagePrice (877ms)
    ✓ long position.averagePrice, buyPrice < averagePrice + minProfitBasisPoints (981ms)
    ✓ short position.averagePrice, buyPrice == averagePrice (800ms)
    ✓ short position.averagePrice, buyPrice > averagePrice (1374ms)
    ✓ short position.averagePrice, buyPrice < averagePrice (1474ms)
    ✓ short position.averagePrice, buyPrice < averagePrice - minProfitBasisPoints (1662ms)
    ✓ long position.averagePrice, buyPrice < averagePrice (908ms)

  Vault.buyUSDG
buyUSDG gas used 352923
    ✓ buyUSDG (454ms)
    ✓ buyUSDG allows gov to mint (316ms)
    ✓ buyUSDG uses min price (345ms)
    ✓ buyUSDG updates fees (963ms)
    ✓ buyUSDG uses mintBurnFeeBasisPoints (292ms)
    ✓ buyUSDG adjusts for decimals (327ms)

  Vault.closeLongPosition
decreasePosition gas used 308308
    ✓ close long position (1180ms)
decreasePosition gas used 307658
    ✓ close long position with loss (1142ms)

  Vault.closeShortPosition
decreasePosition gas used 303984
    ✓ close short position (882ms)
decreasePosition gas used 304722
    ✓ close short position with loss (918ms)

  Vault.decreaseLongPosition
decreasePosition gas used 417540
    ✓ decreasePosition long (2702ms)
    ✓ decreasePosition long aum (1932ms)
    ✓ decreasePosition long minProfitBasisPoints (986ms)
decreasePosition gas used 349322
    ✓ decreasePosition long with loss (1616ms)

  Vault.decreaseShortPosition
decreasePosition gas used 409636
    ✓ decreasePosition short (2504ms)
    ✓ decreasePosition short minProfitBasisPoints (1372ms)
    ✓ decreasePosition short with loss (2423ms)

  Vault.depositCollateral
increasePosition gas used 482793
deposit collateral gas used 284917
    ✓ deposit collateral (5243ms)

  Vault.settings
    ✓ directPoolDeposit (222ms)

  Vault.fundingRates
decreasePosition gas used 415140
withdraw collateral gas used 422531
    ✓ funding rate (1861ms)

  Vault.getFeeBasisPoints
    ✓ getFeeBasisPoints (2492ms)

  Vault.getPrice
    ✓ getPrice (579ms)
    ✓ includes AMM price (944ms)

  Vault.increaseLongPosition
    ✓ increasePosition long validations (1875ms)
increasePosition gas used 482793
    ✓ increasePosition long (1957ms)
increasePosition gas used 482881
    ✓ increasePosition long aum (1745ms)

  Vault.increaseShortPosition
    ✓ increasePosition short validations (1706ms)
increasePosition gas used 476688
    ✓ increasePosition short (5799ms)

  Vault.liquidateLongPosition
liquidatePosition gas used 310465
    ✓ liquidate long (2319ms)
liquidatePosition gas used 348843
    ✓ automatic stop-loss (3278ms)
liquidatePosition gas used 307952
    ✓ excludes AMM price (1452ms)

  Vault.liquidateShortPosition
liquidatePosition gas used 302069
    ✓ liquidate short (2356ms)
liquidatePosition gas used 345720
    ✓ automatic stop-loss (2733ms)
liquidatePosition gas used 299565
    ✓ global AUM (2396ms)

  Vault.sellUSDG
sellUSDG gas used 262912
    ✓ sellUSDG (1025ms)
    ✓ sellUSDG after a price increase (822ms)
    ✓ sellUSDG redeem based on price (835ms)
    ✓ sellUSDG for stableTokens (1193ms)

  Vault.settings
    ✓ inits (89ms)
    ✓ setVaultUtils (71ms)
    ✓ setMaxGlobalShortSize (129ms)
    ✓ setInManagerMode (59ms)
    ✓ setManager (74ms)
    ✓ setInPrivateLiquidationMode (60ms)
    ✓ setIsSwapEnabled (72ms)
    ✓ setIsLeverageEnabled (65ms)
    ✓ setMaxGasPrice (72ms)
    ✓ setGov (410ms)
    ✓ setPriceFeed (93ms)
    ✓ setMaxLeverage (100ms)
    ✓ setBufferAmount (68ms)
    ✓ setFees (213ms)
    ✓ setFundingRate (253ms)
    ✓ setTokenConfig (596ms)
    ✓ clearTokenConfig (381ms)
    ✓ addRouter
    ✓ removeRouter (68ms)
    ✓ setUsdgAmount (310ms)
    ✓ upgradeVault (111ms)
    ✓ setErrorController (209ms)

  Vault.swap
swap gas used 316488
    ✓ swap (2024ms)
    ✓ caps max USDG amount (1005ms)
    ✓ does not cap max USDG debt (678ms)
    ✓ ensures poolAmount >= buffer (677ms)

  Vault.withdrawCollateral
decreasePosition gas used 415140
withdraw collateral gas used 340951
    ✓ withdraw collateral (1592ms)
    ✓ withdraw during cooldown duration (1469ms)
```

```
    ✓ withdraw collateral long (2953ms)
    ✓ withdraw collateral short (4056ms)

  Vault.withdrawFees
    ✓ withdrawFees (970ms)
    ✓ withdrawFees using timelock (1066ms)

  GMT
    ✓ inits
    ✓ setGov (80ms)
    ✓ addAdmin
    ✓ removeAdmin (67ms)
    ✓ setNextMigrationTime (63ms)
    ✓ beginMigration (189ms)
    ✓ addBlockedRecipient (38ms)
    ✓ removeBlockedRecipient (56ms)
    ✓ addMsgSender
    ✓ removeMsgSender (53ms)
    ✓ withdrawToken (237ms)
    ✓ transfer (57ms)
    ✓ approve
    ✓ transferFrom (121ms)
    ✓ allows migrations (183ms)

  Treasury
    ✓ initialize (81ms)
    ✓ setGov (65ms)
    ✓ setFund (50ms)
    ✓ extendUnlockTime (47ms)
    ✓ addWhitelists (93ms)
    ✓ removeWhitelists (117ms)
    ✓ updateWhitelist (140ms)
    ✓ swap (448ms)
    ✓ validates swap.busdSlotCap (353ms)
    ✓ validates swap.busdHardCap (205ms)
    ✓ validates swap.isSwapActive (193ms)
    ✓ addLiquidity (282ms)
    ✓ withdrawToken (92ms)
    ✓ increaseBusdBasisPoints (46ms)
    ✓ endSwap (43ms)

  FastPriceFeed
    ✓ inits (110ms)
    ✓ setTokenManager (62ms)
    ✓ setSigner (74ms)
    ✓ setUpdater (66ms)
    ✓ setFastPriceEvents (56ms)
    ✓ setPriceDuration (84ms)
    ✓ setMinBlockInterval (56ms)
    ✓ setIsSpreadEnabled (81ms)
    ✓ setMaxTimeDeviation (66ms)
    ✓ setLastUpdatedAt (61ms)
    ✓ setVolBasisPoints (63ms)
    ✓ setMaxDeviationBasisPoints (70ms)
    ✓ setMinAuthorizations (67ms)
    ✓ setPrices (260ms)
    ✓ favorFastPrice (320ms)
    ✓ getPrice (750ms)
    ✓ setTokens (191ms)
    ✓ setCompactedPrices (1560ms)
    ✓ setPricesWithBits (1654ms)

  BatchSender
    ✓ setHandler (241ms)

  GmxTimelock
    ✓ inits (181ms)
    ✓ setTokenConfig (447ms)
    ✓ setBuffer (182ms)
    ✓ setIsAmmEnabled (59ms)
    ✓ setMaxStrictPriceDeviation (66ms)
    ✓ setPriceSampleSpace (66ms)
    ✓ setVaultUtils (66ms)
    ✓ setIsSwapEnabled (74ms)
    ✓ setContractHandler (69ms)
    ✓ setIsLeverageEnabled (200ms)
    ✓ setMaxGlobalShortSize (75ms)
    ✓ setMaxGasPrice (83ms)
    ✓ setMaxLeverage (100ms)
    ✓ setFundingRate (256ms)
    ✓ transferIn (136ms)
    ✓ approve (634ms)
    ✓ processMint (572ms)
    ✓ setGov (401ms)
    ✓ setPriceFeed (400ms)
    ✓ withdrawToken (461ms)
    ✓ vaultSetTokenConfig (459ms)
    ✓ priceFeedSetTokenConfig (398ms)
    ✓ addPlugin (273ms)
    ✓ addExcludedToken (119ms)
    ✓ setInPrivateTransferMode (456ms)
    ✓ setAdmin
    ✓ setExternalAdmin (201ms)
    ✓ setInPrivateLiquidationMode (78ms)
    ✓ setLiquidator (199ms)
    ✓ redeemUsdg (861ms)

  OrderBookReader
    ✓ getIncreaseOrders (173ms)
    ✓ getDecreaseOrders (113ms)
    ✓ getSwapOrders (138ms)

  Reader
    ✓ getVaultTokenInfo (252ms)

  Timelock
    ✓ inits (168ms)
    ✓ setTokenConfig (488ms)
    ✓ setBuffer (213ms)
    ✓ mint (286ms)
    ✓ setIsAmmEnabled (56ms)
    ✓ setMaxStrictPriceDeviation (113ms)
    ✓ setPriceSampleSpace (72ms)
    ✓ setVaultUtils (58ms)
    ✓ setIsSwapEnabled (66ms)
    ✓ setContractHandler (57ms)
    ✓ setIsLeverageEnabled (156ms)
    ✓ setMaxGlobalShortSize (70ms)
    ✓ setMaxGasPrice (113ms)
    ✓ setMaxLeverage (113ms)
    ✓ setFundingRate (244ms)
    ✓ transferIn (180ms)
    ✓ approve (595ms)
    ✓ setPriceFeedWatcher (291ms)
    ✓ processMint (599ms)
    ✓ setHandler (610ms)
    ✓ setGov (455ms)
    ✓ setPriceFeed (414ms)
    ✓ withdrawToken (523ms)
    ✓ vaultSetTokenConfig (474ms)
    ✓ priceFeedSetTokenConfig (523ms)
    ✓ addPlugin (401ms)
    ✓ addExcludedToken (166ms)
    ✓ setInPrivateTransferMode (1498ms)
    ✓ batchSetBonusRewards (213ms)
    ✓ managedSetMinter (156ms)
    ✓ managedSetHandler (157ms)
    ✓ setAdmin (55ms)
    ✓ setExternalAdmin (140ms)
    ✓ setShouldToggleIsLeverageEnabled (112ms)
    ✓ setMarginFeeBasisPoints (120ms)
    ✓ setFees (377ms)
    ✓ setSwapFees (334ms)
```

```
    ✓ toggle leverage (354ms)
    ✓ setInPrivateLiquidationMode (110ms)
    ✓ setLiquidator (247ms)
    ✓ redeemUsdg (831ms)

  ReferralStorage
    ✓ Sets new handler (88ms)
    ✓ setTier (81ms)
    ✓ setReferrerTier
    ✓ setReferrerDiscountShare (67ms)
    ✓ setTraderReferralCode (102ms)
    ✓ Registers code (92ms)
    ✓ setCodeOwner (90ms)
    ✓ govSetCodeOwner (95ms)
    ✓ getTraderReferralInfo (86ms)
    ✓ timelock.setTier (84ms)
    ✓ timelock.setReferrerTier (102ms)
    ✓ timelock.govSetCodeOwner (88ms)

  BonusDistributor
    ✓ distributes bonus (793ms)

  RewardRouter
    ✓ inits (76ms)
compound gas used 722503
batchCompoundForAccounts gas used 1000843
    ✓ stakeGmxForAccount, stakeGmx, stakeEsGmx, unstakeGmx, unstakeEsGmx, claimEsGmx, claimFees, compound, batchCompoundForAccounts (3764ms)
mintAndStakeGlp gas used 1096621
unstakeAndRedeemGlp gas used 802620
compound gas used 692849
batchCompoundForAccounts gas used 802620
    ✓ mintAndStakeGlp, unstakeAndRedeemGlp, compound, batchCompoundForAccounts (4223ms)
    ✓ mintAndStakeGlpETH, unstakeAndRedeemGlpETH (1861ms)

  RewardRouterV2
    ✓ inits (206ms)
compound gas used 722917
batchCompoundForAccounts gas used 1001407
    ✓ stakeGmxForAccount, stakeGmx, stakeEsGmx, unstakeGmx, unstakeEsGmx, claimEsGmx, claimFees, compound, batchCompoundForAccounts (3640ms)
mintAndStakeGlp gas used 1096665
unstakeAndRedeemGlp gas used 802620
compound gas used 693263
batchCompoundForAccounts gas used 802620
    ✓ mintAndStakeGlp, unstakeAndRedeemGlp, compound, batchCompoundForAccounts (3345ms)
    ✓ mintAndStakeGlpETH, unstakeAndRedeemGlpETH (2826ms)
    ✓ gmx: signalTransfer, acceptTransfer (3044ms)
    ✓ gmx, glp: signalTransfer, acceptTransfer (9834ms)
    ✓ handleRewards (2898ms)
    ✓ StakedGlp (2562ms)
    ✓ FeeGlp (1450ms)

  RewardTracker
    ✓ inits (62ms)
    ✓ setDepositToken (104ms)
    ✓ setInPrivateTransferMode (78ms)
    ✓ setInPrivateStakingMode (54ms)
    ✓ setHandler (80ms)
    ✓ withdrawToken (83ms)
    ✓ stake, unstake, claim (2045ms)
    ✓ stakeForAccount, unstakeForAccount, claimForAccount (824ms)

  Vester
    ✓ inits (146ms)
    ✓ setTransferredAverageStakedAmounts (150ms)
    ✓ setTransferredCumulativeRewards (133ms)
    ✓ setCumulativeRewardDeductions (161ms)
    ✓ setBonusRewards (140ms)
    ✓ deposit, claim, withdraw (1586ms)
    ✓ depositForAccount, claimForAccount (2222ms)
    ✓ handles multiple deposits (1643ms)
    ✓ handles pairing (3300ms)
    ✓ handles existing pair tokens (2971ms)

  Bridge
    ✓ wrap, unwrap (421ms)
    ✓ withdrawToken (158ms)

  TimeDistributor
    ✓ distribute (439ms)

  USDG
    ✓ addVault (66ms)
    ✓ removeVault (103ms)
    ✓ mint (139ms)
    ✓ burn (127ms)

  YieldFarm
    ✓ stake (128ms)
    ✓ unstake (141ms)

  YieldToken
tranfer0 gas used 63151
tranfer1 gas used 142481
tranfer2 gas used 309173
tranfer3 gas used 155311
    ✓ claim (614ms)
    ✓ nonStakingAccounts (1019ms)


  400 passing (15m)
```

Code Coverage

The core part of the contracts are well covered (under `core/`) with branch coverage > 90%, while not all of the contracts are covered as thoroughly. We recommend adding tests for higher coverage.

The following are the steps to compute the test coverage:

1. Create an `env.json` file with empty values

```
{
    "BSC_URL": "",
    "BSC_DEPLOY_KEY": "",
    "BSCSCAN_API_KEY": "",
    "POLYGONSCAN_API_KEY": "",
    "SNOWTRACE_API_KEY": "",
    "ARBISCAN_API_KEY": "",
    "ETHERSCAN_API_KEY": "",
    "BSC_TESTNET_URL": "",
    "BSC_TESTNET_DEPLOY_KEY": "",
    "ARBITRUM_TESTNET_DEPLOY_KEY": "",
    "ARBITRUM_TESTNET_URL": "",
    "ARBITRUM_DEPLOY_KEY": "",
    "ARBITRUM_URL": "",
    "AVAX_DEPLOY_KEY": "",
    "AVAX_URL": "",
    "POLYGON_DEPLOY_KEY": "",
    "POLYGON_URL": "",
    "MAINNET_URL": "",
    "MAINNET_DEPLOY_KEY": ""
}
```

2. Increase the default balance for the test accounts. Set the `hardhat` network as the following in the `hardhat.config.js`

```
hardhat: {
      allowUnlimitedContractSize: true,
      accounts: {
        mnemonic: "test test test test test test test test test test test junk",
        path: "m/44'/60'/0'/0",
        initialIndex: 0,
        count: 20,
        accountsBalance: "10000000000000000000000000000000000000",
        passphrase: "",
      },
    },
```

3. Add `require("solidity-coverage")` in the `hardhat.config.js`.

4. Run `npm install --save-dev solidity-coverage` to install `solidity-coverage` package.

5. Finally, run `npx hardhat coverage` to get the coverage result.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
access/	100	96.43	100	100	
Governable.sol	100	100	100	100	
TokenManager.sol	100	96.15	100	100	
access/interfaces/	100	100	100	100	
IAdmin.sol	100	100	100	100	
amm/	51.85	16.67	36.36	53.85	
PancakeFactory.sol	0	0	0	0	… 26,28,29,31
PancakePair.sol	100	100	100	100	
PancakeRouter.sol	100	50	100	100	
UniFactory.sol	100	100	100	100	
UniNftManager.sol	0	100	0	0	32
UniPool.sol	0	100	0	0	33
amm/interfaces/	**100**	**100**	**100**	**100**	
IPancakeFactory.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
IPancakePair.sol	100	100	100	100	
IPancakeRouter.sol	100	100	100	100	
core/	95.01	85.91	95.86	95.63	
BasePositionManager.sol	98	90.91	100	97.96	227,243
GlpManager.sol	92.41	83.33	93.33	93.59	… 106,107,125
OrderBook.sol	100	94.68	100	100	
PositionManager.sol	100	92.11	100	100	
PositionRouter.sol	96.84	91.11	96	97.45	548,549,571,589
Router.sol	77.38	58.7	82.14	77.65	… 159,169,195
Vault.sol	98.05	92.86	100	99.02	… 9,1208,1209
VaultErrorController.sol	100	100	100	100	
VaultPriceFeed.sol	80.14	69.15	84	81.56	… 343,365,366
VaultUtils.sol	98.65	97.06	100	98.48	81
core/interfaces/	100	100	100	100	
IBasePositionManager.sol	100	100	100	100	
IGlpManager.sol	100	100	100	100	
IOrderBook.sol	100	100	100	100	
IPositionRouter.sol	100	100	100	100	
IRouter.sol	100	100	100	100	
IVault.sol	100	100	100	100	
IVaultPriceFeed.sol	100	100	100	100	
IVaultUtils.sol	100	100	100	100	
gambit-token/	100	88	100	100	
GMT.sol	100	79.17	100	100	
Treasury.sol	100	96.15	100	100	
gambit-token/interfaces/	100	100	100	100	
IGMT.sol	100	100	100	100	
gmx/	0	0	6.25	0	
EsGMX.sol	0	100	50	0	12
GLP.sol	0	100	50	0	12
GMX.sol	0	100	50	0	12
GmxFloor.sol	0	0	0	0	… 109,113,114
GmxIou.sol	0	0	0	0	… 60,62,63,64
GmxMigrator.sol	0	0	0	0	… 235,236,237
MigrationHandler.sol	0	0	0	0	… 153,155,157
gmx/interfaces/	100	100	100	100	
IAmmRouter.sol	100	100	100	100	
IGmxIou.sol	100	100	100	100	
IGmxMigrator.sol	100	100	100	100	
libraries/GSN/	50	100	50	33.33	
Context.sol	50	100	50	33.33	21,22
libraries/access/	0	0	0	0	
Ownable.sol	0	0	0	0	… 56,64,65,66
libraries/introspection/	75	50	66.67	75	
ERC165.sol	**75**	**50**	**66.67**	**75**	**36**
IERC165.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
libraries/math/	53.33	43.75	50	55.17	
Math.sol	0	0	0	0	… 17,18,19,22
SafeMath.sol	84.21	58.33	75	84.21	140,156,157
UQ112x112.sol	0	100	0	0	15,20
libraries/token/	9.62	11.11	12.5	9.62	
ERC20.sol	0	0	0	0	… 276,277,288
IERC20.sol	100	100	100	100	
SafeERC20.sol	45.45	33.33	50	45.45	… 49,50,54,55
libraries/token/ERC721/	48.72	27.5	45.16	48.1	
ERC721.sol	48.72	27.5	45.16	48.1	… 445,452,453
IERC721.sol	100	100	100	100	
IERC721Enumerable.sol	100	100	100	100	
IERC721Metadata.sol	100	100	100	100	
IERC721Receiver.sol	100	100	100	100	
libraries/utils/	47.12	38.24	46.51	48.18	
Address.sol	54.17	43.75	45.45	57.69	… 167,168,185
EnumerableMap.sol	41.18	37.5	42.86	40	… 216,217,228
EnumerableSet.sol	62.07	33.33	46.67	63.33	… 186,220,241
ReentrancyGuard.sol	100	50	100	100	
Strings.sol	0	0	0	0	… 28,29,30,32
oracle/	94.33	84.29	94.59	94.93	
FastPriceEvents.sol	100	50	100	100	
FastPriceFeed.sol	95.28	89.06	96.55	95.97	… 219,220,315
PriceFeed.sol	81.82	25	83.33	81.82	26,27
oracle/interfaces/	100	100	100	100	
IChainlinkFlags.sol	100	100	100	100	
IFastPriceEvents.sol	100	100	100	100	
IFastPriceFeed.sol	100	100	100	100	
IPriceFeed.sol	100	100	100	100	
ISecondaryPriceFeed.sol	100	100	100	100	
peripherals/	54.87	58.78	75.15	55.2	
BalanceUpdater.sol	0	100	0	0	… 24,25,26,28
BatchSender.sol	90	100	100	90.91	50
EsGmxBatchSender.sol	100	50	100	94.44	47
GmxTimelock.sol	88.74	61.67	82.76	88.46	… 241,245,271
OrderBookReader.sol	100	100	100	100	
Reader.sol	7.83	3.85	5.26	7.66	… 391,392,396
RewardReader.sol	0	100	0	0	… 52,53,54,56
Timelock.sol	95.65	86.54	89.19	95.77	… 326,330,343
VaultReader.sol	0	0	0	0	… 69,70,71,74
peripherals/interfaces/	100	100	100	100	
IGmxTimelock.sol	100	100	100	100	
IHandlerTarget.sol	100	100	100	100	
ITimelock.sol	100	100	100	100	
ITimelockTarget.sol	**100**	**100**	**100**	**100**	
referrals/	87.8	95	92.31	88.1	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
ReferralReader.sol	0	100	0	0	9,11,12,13,16
ReferralStorage.sol	100	95	100	100	
referrals/interfaces/	100	100	100	100	
IReferralStorage.sol	100	100	100	100	
staking/	92.33	71.37	82.58	93.03	
BonusDistributor.sol	86.21	75	66.67	85.71	44,49,64,65
GlpBalance.sol	84.21	60	71.43	84.21	28,37,38
RewardDistributor.sol	92.31	66.67	87.5	96	45
RewardManager.sol	100	100	100	100	
RewardRouter.sol	89.9	76.92	82.61	90	… 183,184,206
RewardRouterV2.sol	96.09	65.63	88.89	96.13	… 217,239,321
RewardTracker.sol	88.89	74	87.5	90.27	… 226,228,230
StakeManager.sol	0	100	0	0	15
StakedGlp.sol	88.46	60	66.67	88.46	45,66,70
Vester.sol	93.79	81.82	83.33	95.14	… 252,257,262
staking/interfaces/	100	100	100	100	
IRewardDistributor.sol	100	100	100	100	
IRewardTracker.sol	100	100	100	100	
IVester.sol	100	100	100	100	
tokens/	57.18	42.5	50.92	57.68	
BaseToken.sol	61.25	50	55.56	60.98	… 202,219,220
Bridge.sol	100	100	100	100	
FaucetToken.sol	0	0	0	0	… 317,328,348
MintableBaseToken.sol	100	50	100	100	
SnapshotToken.sol	0	100	0	0	12,13,14,15
TimeDistributor.sol	91.3	70	91.67	93.18	43,70,71
Token.sol	81.25	50	69.57	81.25	… 207,208,305
USDG.sol	100	100	100	100	
WETH.sol	0	0	0	0	… 289,300,320
YieldFarm.sol	100	100	100	100	
YieldToken.sol	75.64	52.94	66.67	76.25	… 175,186,199
YieldTracker.sol	73.68	57.14	55.56	74.36	… 77,78,79,80
tokens/interfaces/	100	100	100	100	
IBaseToken.sol	100	100	100	100	
IBridge.sol	100	100	100	100	
IDistributor.sol	100	100	100	100	
IGLP.sol	100	100	100	100	
IMintable.sol	100	100	100	100	
IUSDG.sol	100	100	100	100	
IWETH.sol	100	100	100	100	
IYieldToken.sol	100	100	100	100	
IYieldTracker.sol	100	100	100	100	
All files	**75.44**	**67.59**	**72.37**	**75.64**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
7da6eeca142be3e4fbaa9419c6ebab31f99b0012bfec34eb19c8b624dd4c18f7 ./RewardTracker.sol
801392d6a8ddcc84c6f90252fb85b7d2f7fd5fbc58b5a5644a263bdb4cfece5b ./Reader.sol
49a4313eb16685da099d0a25618eecbbf04c0aa264bf89370ed059c51d83b9e0 ./GMX.sol
602279cd37f50ec4c834087230be92880a5f643b0f087992fe65a34a1fc7d21c ./Timelock.sol
c1205fdf682164ce999d7245434772a6155f87ce6f504592270fe695e4acea8f ./Vault.sol
d7728fc4f3215bcb6f96b8c26b8673b3f34353480c76841bc92add39ac7ce71a ./StakedGlp.sol
2654cdc6bbc01e2f16e26924ce36adca3926fabc274774c960a51a4a768a668f ./BasePositionManager.sol
dfd7562d3a237a93249b1cdc8d54eb8b43f2b8f14e0b26bd6fa347ab6a04eab1 ./Router.sol
a6810ef2b2d7abb25abacde3c3ca6a9d9aae2b3f78e048a5f33da10b072650f9 ./RewardRouterV2.sol
30290dcdc924bbf00f36dbf68708e007bc35d84f903f562291aa7394ae09838f ./GlpManager.sol
600bca42435444ec53cf9de9cfd4ce6485d52e31fc4cedda5a734613973981d0 ./OrderBookReader.sol
1ecf38e273bd3ede04444f84dcbe408b8a3f8c3c16cf394a260f39dbadde993d ./PositionRouter.sol
0de67c75f0979c133d0e4d5d55c6ce5f12e5c297c6db0cfa7fe2fbb5a19945ee ./OrderBook.sol
8ab2615c6da4eca477db900d57a300e41042076b2109168e9ad53b3b151e7253 ./PositionManager.sol
```

Changelog

- 2022-09-12 - Initial report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.