GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

GMX

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, 0xKato, Kristian Apostolov, Deliriusz

Client Firm GMX

Final Report Date Preliminary Report

Audit Summary

GMX engaged Guardian to review the security of its decentralized synthetics perpetuals exchange. From the 22nd of June to the 11th of July, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Arbitrum, Avalanche**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/GMX_3

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics/
Commit(s)	fa95bcffa8a29ccf42acc228bcaabf47ae5467ab

Audit Summary

Delivery Date	Preliminary Report
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	1	0	0	0	0
High	3	3	0	0	0	0
Medium	16	16	0	0	0	0
Low	28	28	0	0	0	0

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ADLU	AdlUtils.sol	4bf3ea9b168bbd1bd61d8ae8583145b342b867ba
BNK	Bank.sol	661c2f7e4227e315febf5800510a25a77f16bb16
SBNK	StrictBank.sol	13276745295cbc093207e92bcb096c9a01e79c99
CBKU	CallbackUtils.sol	2bb0ad384337fbbf690f9493bd55100c8bb3b9e4
DCBK	IDepositCallbackReceiver.sol	7b53a4c8082957b0f7f6aa0cc3e20d21cb1e3605
OCBK	IOrderCallbackReceiver.sol	156914da44b29805e1a5c9d5dca8160403048222
WCBK	IWithdrawalCallbackReceiver.sol	9e114c4b16376182ca7e3708c9efd708fe4d3061
ARBS	ArbSys.sol	0d9703a3477e40ccce9b0b526c0c9f4310034496
CHAIN	Chain.sol	9c435faa3ba666b16fa2054c0b39e01aa030d0a0
DATA	DataStore.sol	93a8457b50afd9dcbd1e52c7efc372c345d68971
KEY	Keys.sol	65e494a4336ef74bc632430974300976439d0b9b
DEP	Deposit.sol	9580b364ab6c14e76db4a0058a785cf10264c241
DEPS	DepositStoreUtils.sol	f6c25343cf7e26d14236ef72c25dd596d7fc30fa
EMIT	EventEmitter.sol	3fce680d9fd7432923b859ab7f9fa15e8a96ee14
DEPH	DepositHandler.sol	8228d14c7e7f1b1849ed939cae754b4b2d143499
ORDH	OrderHandler.sol	e26659e15b627b97b12057c7d874e69acc406e77
WTDH	WithdrawalHandler.sol	b87a2da08be176ffd1523148ecefa227cadb84a3
FTU	FeatureUtils.sol	4fea0cf326251322102df2ffe74c6bb663a03246
ODV	OrderVault.sol	74f991769825ba9fc8b98f3be3a5fefc32be7539
DPV	DepositVault.sol	1d19ad5afc0baec27a608a2f53cbb5b6f48f8f26

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
WDV	WithdrawalVault.sol	5cc2b331b13f735dfebc983b9aec705692e0d2a2
GSU	GasUtils.sol	e81262df819ff2a1e421de2e84fc93d5ebfca849
LIQU	LiquidationUtils.sol	740b9b5aeaad27924a1a0c72b229b6dede0007e8
MKT	Market.sol	a66a2a9127674ffb23d74d7a252f046f98c2e182
MKTF	MarketFactory.sol	b1418f56e89d4526739010caf69b937202301529
MKTS	MarketStoreUtils.sol	5f938ae585a9541bbf4f8c3561edb442694a3f46
MKTT	MarketToken.sol	a55f9a9931d906583050b4f01b74b7adbe54cf1d
MKTU	MarketUtils.sol	10a4d7ed01b9b65d8f0bc69f7c65ef70ee289153
NCU	NonceUtils.sol	6ec2082417987d5c4e859adefb9b28efb1ed5c39
PFE	IPriceFeed.sol	431babdd9ab4ee30ae9eba84f469620a3d2951f3
OCL	Oracle.sol	4771d33d54aaef1a8fe3061529d39fef8ac53f00
OCLM	OracleModule.sol	6361bd07eea14b864fce8e88ab0592b6b0e82674
OCLS	OracleStore.sol	5b87b5af1af681ee020fb0c65ddd4f184c9bef39
OCLU	OracleUtils.sol	4f386e1fc0205d5cc7cfd1dd5213f86fe78774fa
DOU	DecreaseOrderUtils.sol	a05eedf4d4b916c82ee3f76ef6f975743b7808b4
IOU	IncreaseOrderUtils.sol	a9c462ff258d7f72fdf0c337460750298709a4a5
ORD	Order.sol	fe296c4e1cba04e370a9fe576de61512bfb45b1b
BOU	BaseOrderUtils.sol	3cede11449aa54c89646c5e73af04eeab6f4c7bc
ORDS	OrderStoreUtils.sol	40ece421c2de62b0812b17b9409008cd31d8a45f
FEU	FeeUtils.sol	419e95c99abe2bfa0fafc872eff2a451cfde9740

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ORDU	OrderUtils.sol	726ce70fd9f6fda1ca60291fb5dc9888880d34de
SWOU	SwapOrderUtils.sol	856262e9af4e709f9c433e3096d914ace4fb8c1a
DPU	DecreasePositionUtils.sol	582a9a0623226d983680fd3237be08e0271512c6
IPU	IncreasePositionUtils.sol	0b203cb45b6ab3374a1a0d5043f15047dfdb3ff6
POS	Position.sol	73596d9de7117c3c44c176ac0d2fc9627743ff9e
POSU	PositionUtils.sol	a5c87aae2b1487e7e90d745b5ffe3bf6dd51f5cf
PRICE	Price.sol	c1f87807a20c43c1710d1e3c3e628e265cd5686e
PPU	PositionPricingUtils.sol	ef8e456e9c7272b9da8a0cada726ba2172794e10
PRU	PricingUtils.sol	a1be554641c1b75b6c321baca4c9e5c722e05a53
SPRU	SwapPricingUtils.sol	5c094c9a4c742e164f38a56ea7bb83049a1b4866
READ	Reader.sol	12343e67be606e67b69ca4c8da7f9a9d6e24745e
IREFS	IReferralStorage.sol	f61d9bb3c2ec803d3b97c1e7f4faca4f1e517bf6
REFT	ReferralTier.sol	8a34d5e24b6a317b063ebd59d85fa1fec9307ea7
REFU	ReferralUtils.sol	0f761d11b0853d86db22f1a2015f4f0e03aec89b
REFS	ReferralStorage.sol	086c0102b673a95198c213003ba1e0882dbd6a87
ROLE	Role.sol	86935a3af0c782e711076d1a2ad2222bda7185fa
ROLEM	RoleModule.sol	6ff5de5a0bea585ad4195784a9f3d2013cdb935d
ROLES	RoleStore.sol	6717a28a2dc4f77505edf1a6989a559405a86883
DEPU	DepositUtils.sol	31d78e6d324c3af1d8fb65f75bda4d5d88498be2
EDPU	ExecuteDepositUtils.sol	ad21b611c2c620fe25c5743c1d2d7f5766488430

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ERTR	ExchangeRouter.sol	2d48d92db5676c66fe2ecf7c456a7b184bcf4f79
RTR	Router.sol	0fde38bae3c62565cda7fec0ba521a46611d6e32
SWPH	SwapHandler.sol	9e3bb4bb999a70390ff2be5f447a7d4ffd5699c5
SWPU	SwapUtils.sol	02cc683fcba9ef23a8de7933a43914734f0a91a2
TIME	Timelock.sol	80359b9e696224ec3d63ab8a557548a7015ddac8
IWNT	IWNT.sol	972554584395e769df3392828d0e43adc74801f4
TU	TokenUtils.sol	dfbaa478edbc1f862cf0649d7c7f91debb82db1b
ARR	Array.sol	a27f1de5a45f6fd95f9a58f2e2a39df22208f7ff
BIT	Bits.sol	c7fa3c25af05c172cff6faccef14182665b875ba
CLC	Calc.sol	ffc7e4f0e4908afd72468dde47b7e9f7e7e3c1c4
ENM	EnumerableValues.sol	36354b53a39c4fb584313f8d3aac8e2b091d90a2
MC	BasicMulticall.sol	c23389da01002c95d775b798ccd850fa463ff6c5
PMC	PayableMulticall.sol	4af36b2f3fba97ab03e201cebb419c8897f5edd1
PREC	Precision.sol	a224e4fddc818c740c6ea87f91989d02a04c187e
WTD	Withdrawal.sol	9400ab833a8ec81c21475f30512256dd0bd5cc66
WTSU	WithdrawalStoreUtils.sol	e1bab1c92a3338dbf85bebaf7046eb2ead479343
WTDU	WithdrawalUtils.sol	17ef60c89e7e1f2e8fc5c7c414df7b60a726a38c
CBU	CallbackUtils.sol	b57c3a07448c6e5d75207ecedd36d924a4ffc575
CON	Config.sol	c47f2abda71bf0874a59aa885bbdadf3c43aa988
ERR	Errors.sol	f022e26738e729b2767192e36d733ac9c9e3e75d

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ERTR	BaseOrderHandler.sol	060f8d1682aa414ce853a5d82140f74eeeb6d67a
EUTL	ExchangeUtils.sol	97af1a3cbb640fa072e259f1785255ae16f96612
ERTR	LiquidationHandler.sol	3851b032bf6178455db147754536658cdd188d60
FEEH	FeeHandler.sol	f8f4e7130f55cf29db963fa40bc2dbb10e485718
MPVI	MarketPoolValueInfo.sol	63ced41c9271ea31c4ff33f1ad734c734098381a
DPSU	DecreasePositionSwapUtils.sol	c93c23fd338ceacf33110b33eb02d13bda3c5cae
PSU	PositionStoreUtils.sol	26bc7ae6b476f9afaf7b6deaf55921427813ae0f
ACC	AccountUtils.sol	2a934679f6138775382c620fd94974f87948748d
CAST	Cast.sol	68780489ad9ee795bf3d0574e96b399d36504f58
GRG	GlobalReentrancyGuard.sol	4f4a5deed4a1f00e7a349f87f5af802b85e8ba3b
MASK	Uint256Mask.sol	d5ec9bd3b5f72c11e8d93b0a4e1275f430cfdfa4
ADLH	AdlHandler.sol	9554308173b469c0b3bb9cba3e17fda56dd9c3aa
DPCU	DecreasePositionCollateralUtils.sol	cab7225c47ddb676525b7b5ea6a44e7a75a5a3f0
ERRU	ErrorUtils.sol	6e1290f8503c73a2a0f96f82d7d975aad22eb231

Audit Scope & Methodology

Vulnerability Classifications

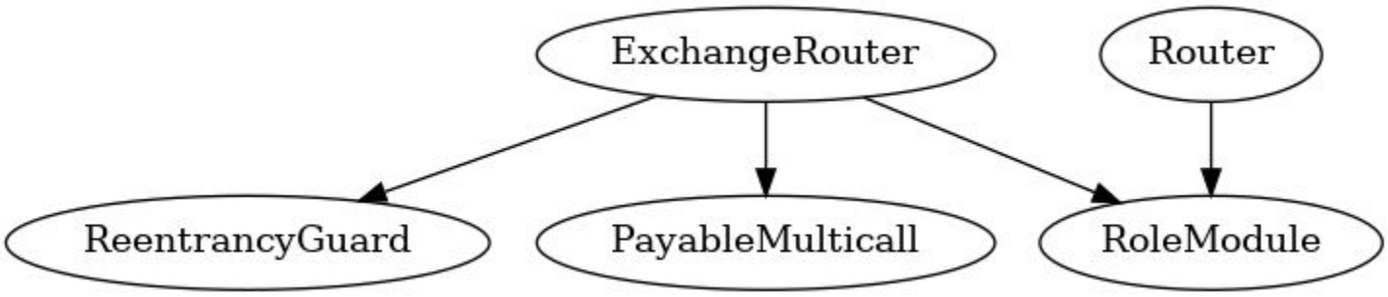
Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract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

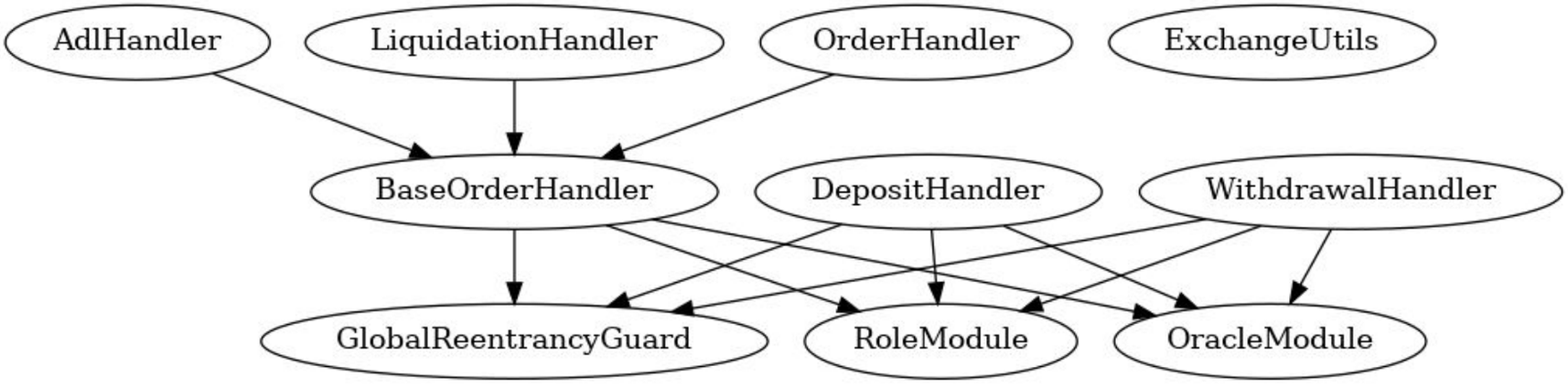
Methodology

The auditing process pays special attention to the following considerations:

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

Inheritance Graph





Findings & Resolutions

ID	Title	Category	Severity	Status
DPCU-1	fundingFeeAmountPerSize Errantly Reset	Logical Error	Critical	Unresolved
POSU-1	willPositionCollateralBeSufficient Validation Bypassed	Validation	High	Unresolved
ORDH-1	Keeper Griefed With orderUpdatedAtBlock	Griefing	High	Unresolved
MKTU-1	Borrowing Fees Avoided Due To Skip	Protocol Manipulation	High	Unresolved
IPU-1	Incongruent Price Impact	Logical Error	Medium	Unresolved
EDPU-1	Invalid Deposit Price Impact For Homogenous Markets	Logical Error	Medium	Unresolved
DPCU-2	priceImpactDiffUsd Paid Before priceImpactUsd	Prioritization	Medium	Unresolved
GSU-1	Decrease Swap Type Not Included In Gas Estimation	Gas Estimation	Medium	Unresolved
EDPU-2	Subsequent Mints Cause Market Token Inflation	Logical Error	Medium	Unresolved
IOU-1	Overwritten Callback Contract	Misconfiguration	Medium	Unresolved
MKTU-2	Negative Pool Value DoS	DoS	Medium	Unresolved
GLOBAL-1	Issues With Equity Synthetic Tokens	Stock Splits	Medium	Unresolved
DPCU-3	Collateral Prioritized Over PnL	Prioritization	Medium	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
MKTU-3	Shorts Arbitrarily Pay Stable Funding To Longs	Incentives	Medium	Unresolved
DPCU-4	Non-zero Effect On Pool Value	Logical Error	Medium	Unresolved
GSU-2	getExecutionGas Needs to Account for Callback Gas	Logical Error	Medium	Unresolved
GLOBAL-2	Saved Callback Keeper Griefing	Gas Griefing	Medium	Unresolved
OCL-1	Chainlink Feed Manipulation	Protocol Manipulation	Medium	Unresolved
MKTU-4	Unwieldy Claimable Collateral Controls	Protocol Manipulation	Medium	Unresolved
GLOBAL-3	Unbounded Virtual Inventory Price Impact	Suggestion	Medium	Unresolved
DPU-1	Duplicate collateralTokenPrice Fetched	Optimization	Low	Unresolved
MKTU-5	Typo	Typo	Low	Unresolved
MKTU-6	Redundant if Case	Optimization	Low	Unresolved
DPCU-5	Lack Of Event Data	Events	Low	Unresolved
DPCU-6	Inefficient Price Fetching	Optimization	Low	Unresolved
DPCU-7	initialCollateralDeltaAmount Unexpected Adjustment	Unexpected Behavior	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-4	Empty Event Data	Events	Low	Unresolved
DPCU-8	Inaccurate Comment	Comments	Low	Unresolved
POSU-2	getPositionPnlUsd No Longer Needs Separate Prices	Superfluous Code	Low	Unresolved
IPU-2	Superfluous if Case	Superfluous Code	Low	Unresolved
GLOBAL-5	Users Negatively Impacted By Price Spread	Documentation	Low	Unresolved
GLOBAL-6	Missing NatSpec	Documentation	Low	Unresolved
MKTU-7	Stable Funding Factor Liquidation Risk	Configuration	Low	Unresolved
ARR-1	Check Odd Gas Optimization	Optimization	Low	Unresolved
GLOBAL-7	Liquidation Fee	Suggestion	Low	Unresolved
IOU-2	Swap Before Updating Borrowing State	Protocol Manipulation	Low	Unresolved
GLOBAL-8	Floating Pragma Version	Best Practices	Low	Unresolved
IPU-3	Unnecessary perSizeValues Initialization	Optimization	Low	Unresolved
SWPU-1	Misleading priceImpactUsd Emitted	Events	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
POSU-3	Outdated Price Impact Formula	Documentation	Low	Unresolved
MKTU-8	Missing swapPath Validation	Validation	Low	Unresolved
DATA-1	Superfluous Stack Variable	Optimization	Low	Unresolved
ERTR-1	ExchangeRouter Inefficient Loops	Optimization	Low	Unresolved
GLOBAL-9	Empty Swap Path For Swap Orders	Optimization	Low	Unresolved
ARR-2	Oracle Block Validation Optimizations	Inconsistency	Low	Unresolved
GLOBAL-10	Price Impact Incongruence	Optimization	Low	Unresolved
GLOBAL-11	Superfluous Price Impact Logic	Optimization	Low	Unresolved
GLOBAL-12	Redundant Price Fetching	Optimization	Low	Unresolved

DPCU-1 | Position fundingFeeAmountPerSize Errantly Reset

Category	Severity	Location	Status
Logical Error	● Critical	DecreasePositionCollateralUtils.sol: 374	Unresolved

Description PoC

When the `fees.totalCostAmountExcludingFunding` is paid with any amount of secondary tokens the `fees` object is replaced with an empty instance.

In this case however there can be a position that remains, and it will be stamped with a `fundingFeePerSize` value of 0 from this empty `fees` instance. Therefore the position must now pay all funding fees since the inception of the market.

In all likelihood the position will then be immediately liquidated leading to an immediate significant loss of assets for the position that would have remained.

Recommendation

Do not reset the `fees.funding.funding.latestFundingFeeAmountPerSize` on the `fees` object when the position can remain, e.g. outside of any insolvent close.

Resolution

POSU-1 | willPositionCollateralBeSufficient Validation Bypassed

Category	Severity	Location	Status
Validation	High	PositionUtils.sol: 415	Unresolved

Description

The willPositionCollateralBeSufficient validation aims to decide whether or not the collateral amount that remains for a position will be sufficient for it's leverage.

However in many cases this validation allows decrease orders which will put the position's collateral below the "sufficient threshold". This is because the willPositionCollateralBeSufficient validation excludes fees that will be subtracted from the position's collateral as well as negative price impact that can be applied to the position's collateral.

Recommendation

Account for fees and potentially even negative price impact in the willPositionCollateralBeSufficient so that the validation cannot be circumvented in these cases.

Resolution

ORDH-1 | Keeper Griefed With orderUpdatedAtBlock

Category	Severity	Location	Status
Griefing	High	OrderHandler.sol: 241	Unresolved

Description

Orders will remain in the order store when the `OracleBlockNumbersAreSmallerThanRequired` error occurs during execution. Therefore a malicious user can cause the keeper to continuously expend gas to attempt to execute an order without requiring any additional `executionFee`.

Consider the following scenario:

- A malicious user frontrun's the keepers execution tx and updates their order, updating the `orderUpdatedAtBlock`.
- Now the keeper's tx goes through all of the price setting and order execution logic up to the oracle block number validation.
- Then the order execution tx reverts, the keeper has spent a significant amount of gas but the order still remains, with the same `executionFee` still attached.
- The malicious user may continue to do this and continue to gas grief the keeper.
- The malicious user can then cancel their order at any time to receive their `executionFee` back.

The scenario can be exacerbated with an order that requires many prices to be set where the malicious user includes a maximum length `swapPath` that requires many prices.

Recommendation

Require a non-refundable WNT payment upon order updates to disincentivize such attacks. Otherwise consider validating the oracle block numbers for the order early on in the order execution to minimize the amount of gas that can be wasted.

Resolution

MKTU-1 | Borrowing Fees Avoided Due To Skip

Category	Severity	Location	Status
Protocol Manipulation	High	MarketUtils.sol: 2145-2160	Unresolved

Description PoC

Users who have accumulated a large amount of borrowing fees can avoid paying these fees as long as the `cumulativeBorrowingFactor` has not yet been incremented and `skipBorrowingFeeForSmallerSide` is enabled.

Consider the following scenario:

- Long OI > Short OI
- Trader A has a large long position that has accumulated a significant amount of borrowing fees. These borrowing fees have not been "solidified" by the update of any other long position. Therefore the `cumulativeBorrowingFactor` has not yet been incremented to account for Trader A's accumulated borrowing fees since the last recorded `cumulativeBorrowingFactorUpdatedAt` for longs.
- Trader A opens another short position shifting the larger side to be the opposite of the first position.
- Trader A then closes the first position and pays 0 borrowing fees.

Recommendation

Do not allow previously accumulated borrowing fees to be skipped in the event that a trader forces a side to have the smaller OI. This can be achieved by updating both the long and short borrowing fees upon position increase and decrease.

Resolution

IPU-1 | Incongruent Price Impact

Category	Severity	Location	Status
Logical Error	Medium	IncreasePositionUtils.sol: 361-369	Unresolved

Description

The price impact amount represented by the executionPrice may not match the price impact amount calculated. This is because the index price used to calculate the price impact amount may differ from the index price used in getExecutionPriceForIncrease.

Consider the following scenario where a trader increases a long position:

```
Price Impact USD = -$100; Index Price = ($50, $100)
```

In IncreasePosition:

```
index price = indexTokenPrice.min = $50
price impact amount = -$100 / $50 = -2 tokens
```

In BaseOrderUtils:

```
index price = indexTokenPrice.pickPriceForPnl(isLong, true) = $100
price impact amount = -$100 / $100 = -1 tokens
```

The discrepancy is because $100 is used for the index price in getExecutionPriceForIncrease rather than $50. As a result, the executionPrice doesn't reflect the price impact amount which is added to the baseSizeDeltaInTokens.

Recommendation

executionPrice can simply be calculated as:

```
executionPrice = params.order.sizeDeltaUsd() / cache.sizeDeltaInTokens
```

to reflect the price impact amount used and then emitted in the event.

EDPU-1 | Invalid Deposit Price Impact For Homogenous Markets

Category	Severity	Location	Status
Logical Error	Medium	ExecuteDepositUtils.sol: 168-179	Unresolved

Description PoC

When depositing, swap impact is calculated for the `longTokenUsd` and `shortTokenUsd` being deposited.

However in the case of markets where `longToken == shortToken`, all the value being deposited will be in the `longTokenUsd`.

The market will always be considered balanced to begin with since the `poolAmount` is simply divided by two for both sides. Therefore every deposit will receive negative impact because each deposit is treated as if it is adding all its value to the long side and therefore unbalancing the pool.

Recommendation

Skip price impact calculations when depositing into markets where `longToken == shortToken`.

Resolution

DPCU-2 | priceImpactDiffUsd Paid Before priceImpactUsd

Category	Severity	Location	Status
Prioritization	Medium	DecreasePositionCollateralUtils.sol: 388, 431	Unresolved

Description

During an insolvent close, the `priceImpactDiffUsd` is paid at a higher priority than the negative price impact.

This means there are scenarios where the position is liquidated or ADL'd and the account is credited with claimable tokens for price impact that was capped, meanwhile the base price impact, that was the uncapped portion, goes unpaid.

Ultimately this benefits the user and hurts the protocol because these funds will become claimable for the user rather than going towards the `positionImpactPool` and `poolAmount` to cover as much of the price impact amount as possible.

Recommendation

Pay the `priceImpactUsd` before the `priceImpactDiffUsd`.

Resolution

GSU-1 | Decrease Swap Type Not Included In Gas Estimation

Category	Severity	Location	Status
Gas Estimation	Medium	GasUtils.sol: 200	Unresolved

Description

When a decrease order contains a `decreasePositionSwapType` other than `NoSwap` it will execute an additional swap in the current market. However this additional swap is not accounted for in the `estimateExecuteDecreaseOrderGasLimit` gas estimation.

Therefore these orders will consume gas for an extra swap that is not accounted for in the estimated gas cost.

Additionally if this extra swap were to be accounted for by default in the base `decreaseOrderGasLimit`, it would be requiring users with a `NoSwap` to put down more initial `executionFee` than necessary.

Recommendation

Account for an additional `gasPerSwap` for decrease orders that have a `decreasePositionSwapType` other than `NoSwap`.

Resolution

EDPU-2 | Subsequent Mints Cause Market Token Inflation

Category	Severity	Location	Status
Logical Error	Medium	ExecuteDepositUtils.sol: 351, 388	Unresolved

Description

In the usdToMarketTokenAmount function when the supply of market tokens is 0 and the poolValue is nonzero, the resulting market token amount is the poolValue + usdValue.

However in the _executeDeposit function, the marketTokensSupply and poolValue variables are cached and passed to the usdToMarketTokenAmount function twice in a row when there is positive impact.

When the supply of market tokens is zero and there is some dust leftover in the poolAmount it is possible to be positively impacted and end up with:

```
2 * poolValue + (positiveImpactAmount.toUint256() * _params.tokenOutPrice.max)
+ (fees.amountAfterFees * params.tokenInPrice.min) market tokens
```

Thus resulting in a market token amount that double counts the poolValue. The case where this market token amount inflation occurs is rare and there is no immediate financial loss or gain.

However, this market token inflation is unexpected and goes against the documented goal of a 1 USD value per market token in the usdToMarketTokenAmount function. This unexpected and undocumented behavior could be used to exploit systems building on top of GMX V2.

Recommendation

Consider re-calculating the updated poolValue and market token supply after the positive impact application if the initial market token supply was 0. Otherwise document this unexpected behavior.

IOU-1 | Overwritten Callback Contract

Category	Severity	Location	Status
Misconfiguration	Medium	IncreaseOrderUtils.sol: 76	Unresolved

Description

The saved callback contract is set anytime an `IncreaseOrder` is processed, therefore the following unexpected scenario may arise:

1) A trader creates a limit increase order for their position without a callback contract.
2) The trader now sends a market increase order with callback contract A to save contract A.
3) The trader's limit increase order then executes.
4) The saved Callback contract is now overwritten with `address(0)`, and upon liquidation or ADL, no callback action occurs.

This may cause unexpected results for the trader considering the callback may be performing a useful action such as closing out a hedge/position on another platform. Additionally, the trader might only want the callback to be executed for their increase, and not on any subsequent liquidation or ADL.

Recommendation

Remove the `setSavedCallbackContract` function from the `IncreaseOrder` flow so it can be set explicitly with a separate configuration function. Additionally, clearly document that there can only be 1 saved callback contract per market e.g. if a user has two positions long and short in the same market both will use the same callback contract.

Resolution

MKTU-2 | Negative Pool Value DoS

Category	Severity	Location	Status
DoS	Medium	MarketUtils.sol: 235	Unresolved

Description

When the `result.poolValue` is negative it is impossible to deposit into a market to make it useable again. There can be leftover index tokens in the position impact pool which are subtracted from the pool value. Consequently, it is possible to achieve a state where the supply of market tokens is 0, but the pool value is negative.

Such a scenario would shut down the market, preventing inflow of any deposits and further usage. However, it is important to note that such a scenario would be rare.

Recommendation

Clearly document that such a scenario can occur, and monitor impact factors to help prevent such a situation from arising.

Resolution

GLOBAL-1 | Issues With Equity Synthetic Tokens

Category	Severity	Location	Status
Stock Splits	Medium	Global	Unresolved

Description

Equities will potentially be supported for trading, as long as they have a price feed. Potential issues arise in the case of forward stock splits, where the price per share is halved and the number of shares a user owns double. This may require an update to the size in tokens for existing positions or bespoke price logic. Other scenarios include reverse stock splits, mergers and acquisitions, etc.

Recommendation

Document protocol behavior in such scenarios and careful monitor markets where such an event is approaching as they are announced in advance.

Resolution

DPCU-3 | Collateral Prioritized Over PnL

Category	Severity	Location	Status
Prioritization	Medium	DecreasePositionCollateralUtils.sol: 588	Unresolved

Description

In the payForCost function, the collateral token is prioritized over the secondary token when making payments. As a result, a user's collateral could be unexpectedly reduced when their is substantial PnL or price impact in the secondary token available to cover the cost.

This can be especially unexpected for costs such as negative price impact which would commonly be thought of as affecting the execution price and being deducted from the PnL.

Recommendation

Consider prioritizing secondary token amounts over collateral when paying amounts such as negative price impact. Otherwise, document that the user's collateral will be prioritized over secondary PnL.

Resolution

MKTU-3 | Shorts Arbitrarily Pay Stable Funding To Longs

Category	Severity	Location	Status
Incentives	Medium	MarketUtils.sol: 1946, 1948	Unresolved

Description

In the event that OI is balanced for longs/shorts, shorts will arbitrarily pay longs because of the definition of `result.longsPayShorts`:

`result.longsPayShorts = cache.longOpenInterest > cache.shortOpenInterest`

Normally, the `fundingUsd` would be 0 in this case as the resulting `fundingFactorPerSecond` is 0 when the OI is balanced. However when there is a stable funding factor configured the `fundingFactorPerSecond` will be nonzero when the OI is balanced.

Therefore when there is a stable funding factor configured, shorts will arbitrarily pay longs the stable funding factor. This causes an incentive to stop shorting and join the long side to collect funding fees rather than pay them. Even though OI is already balanced, which undermines the point of funding fees.

Recommendation

Skip the update per size delta logic when there is a stable funding factor present and the long open interest is equivalent to the short open interest.

Resolution

DPCU-4 | Non-zero Effect On Pool Value

Category	Severity	Location	Status
Logical Error	Medium	DecreasePositionCollateralUtils.sol: 153-180	Unresolved

Description

When the index token is the same as the PnL token, the positive price impact amount deducted from the position impact pool is maximized (round up division & divide by min price) meanwhile the deduction amount for the pool is minimized (round down division & divide by max price).

Similarly, when the index token is the same as the collateral token, the negative price impact amount added to the position impact pool is minimized (multiplied by the min price and divided by the max price) while the pool amount delta is not.

This creates a tendency for positive price impact to err on the side of increasing the pool value as stated in the comment on line 171. Because of the unequal effects on the impact pool amount and the pool amount there is an immediate non-zero impact on the pool value.

This behavior opens up the possibility for market depositors to manipulate price impact and absorb the position impact pool value. Such a manipulation can be straightforward when the index token is the same as either the collateral token or PnL token.

Recommendation

When the index token is the same as the collateral token or the PnL token consider using the same prices to convert usd values to token values during both positive and negative price impact accounting.

Resolution

GSU-2 | getExecutionGas Needs to Account for Callback Gas

Category	Severity	Location	Status
Logical Error	Medium	GasUtils.sol: 42	Unresolved

Description

The `GasUtils.getExecutionGas` function sets aside a `minHandleErrorGas` amount to handle the subsequent error logic. However, this amount does not take into account the configured callback gas limit for an order in the event of a cancellation/freezing.

Recommendation

Include the configured callback gas limit for the order being executed in the `minHandleErrorGas` result.

Resolution

GLOBAL-2 | Saved Callback Keeper Griefing

Category	Severity	Location	Status
Gas Griefing	Medium	Global	Unresolved

Description

Saved callback contracts will be executed on liquidation and ADL orders which are entirely funded by the keeper/protocol without any remuneration from the user. Additionally, the saved callback contract will be given the maximum callback gas limit upon liquidation or ADL.

This way malicious traders may grief the keeper by creating numerous positions that become liquidatable simply to force the keeper to expend the maximum callback gas limit. Notice that a malicious trader may also be able to directly extract value from the keeper with gas tokens upon a liquidation or ADL callback since this execution gas is subsidized by the protocol.

Recommendation

Be aware of the potential for significant uncovered gas expenditure. Consider implementing a mechanism to remunerate the keeper for excessive callback gas expenditure from the user's remaining collateral in the case of liquidation and ADL orders.

Resolution

OCL-1 | Chainlink Feed Manipulation

Category	Severity	Location	Status
Protocol Manipulation	Medium	Oracle.sol: 539	Unresolved

Description

In the event that a configured Chainlink price feed is outdated the order execution transaction will not occur due to a `PriceFeedNotUpdated` revert.

A malicious trader may observe that the price feed is outdated and submit a market order that includes a token requiring that price feed. The trader's order will not be executed as the price feed is outdated.

The trader can then observe that the chainlink prices have been updated with a `transmit` transaction and choose to cancel their order if price has not moved favorably in the time that the price feed was outdated.

Recommendation

Carefully monitor outdated Chainlink price feeds and consider implementing logic to freeze/cancel orders that cannot be executed.

Resolution

MKTU-4 | Unwieldy Claimable Collateral Controls

Category	Severity	Location	Status
Protocol Manipulation	Medium	MarketUtils.sol: 594	Unresolved

Description

In the `claimCollateral` function the `claimableFactor` is the maximum of the `claimableFactorForTime` and `claimableFactorForAccount`. Therefore in the case where capped negative price impact is manipulated and claimable collateral ought to be used at the protocol's discretion to punish the manipulator, the controls will be insufficient.

Consider the following:

- A malicious trader manipulates reference prices to take advantage of negative PI capping.
- Many other traders are capped in the same `timekey` as this one malicious trader.
- The other traders ought to be able to claim their collateral at a later date, but the malicious trader should never have their `claimableFactor` updated.

The current controls are not well suited for this scenario since every non-malicious trader would have to have a manual per-account `claimableFactorForAccount` configured. In the case where there are many innocent traders in this `timekey` this may be impractical, especially in times of market volatility. The controls should be flipped such that the single malicious trader can be punished with a more constrictive `claimableFactorForAccount`.

Recommendation

Alter the `claimableFactorForAccount` logic such that it is a more constrictive threshold rather than a less constrictive one.

This would likely be accompanied by a `isClaimableFactorForAccountEnabled` boolean value to indicate whether the `claimableFactorForAccount` ought to be used in the case that it is the default value of zero.

GLOBAL-3 | Unbounded Virtual Inventory Price Impact

Category	Severity	Location	Status
Suggestion	Medium	Global	Unresolved

Description

There is no lower bound on how negative price impact can be during swaps or position orders due to the virtual inventory. In virtual inventories where there are many markets, it is possible to build up large imbalances over time as users may be incentivized with positive impact from their direct pool to make deposits or orders that ultimately increase the disparity in the virtual inventory.

The resulting extreme disparity in the virtual inventory will lead to significant negative impact without bound for unsuspecting users. The most likely outcome is that deposits and orders that would compute price impact from the virtual inventory will simply be cancelled due to their acceptable price or minimum amounts being unfulfillable.

Recommendation

Consider implementing a lower bound on the magnitude of negative price impact that can be applied from the virtual inventory. Alternatively, consider creating separate price impact factors that can apply specifically to the virtual inventory calculations as the virtual inventory imbalances can be significantly larger than normal markets.

Resolution

DPU-1 | Duplicate collateralTokenPrice Fetched

Category	Severity	Location	Status
Optimization	Low	DecreasePositionUtils.sol: 73, 148	Unresolved

Description

In `decreasePosition` there exists a `cache.collateralTokenPrice` which is stored at the beginning of the function execution on line 73.

However this price is retrieved for a second time with `getCachedTokenPrice` on line 148.

Recommendation

Reuse the `cache.collateralTokenPrice`.

Resolution

MKTU-5 | Typo

Category	Severity	Location	Status
Typo	Low	MarketUtils.sol: 1942	Unresolved

Description

The comment `if there is a stable funding factor then used that instead of the open interest` misspells “use” as “used”.

Recommendation

Replace “used” with “use”.

Resolution

MKTU-6 | Redundant if Case

Category	Severity	Location	Status
Optimization	Low	MarketUtils.sol: 1100, 1112	Unresolved

Description

The same `if` condition relying on `result.longsPayShorts` is repeated back to back. The logic in both can be deduplicated into a single `if` condition.

Recommendation

Consolidate the contents of each `if (result.longsPayShorts)` condition into a single `if` case.

Resolution

DPCU-5 | Lack Of Event Data

Category	Severity	Location	Status
Events	Low	DecreasePositionCollateralUtils.sol: 243	Unresolved

Description

When emitting the `emitInsufficientFundingFeePayment` event, it may be helpful to include the `amountPaidInSecondaryOutputToken` as this is useful when determining how much collateral token needs to be deposited from an insurance fund or other into a market.

Recommendation

Consider adding the `amountPaidInSecondaryOutputToken` as a piece of data emitted with the `emitInsufficientFundingFeePayment` function call.

Resolution

DPCU-6 | Inefficient Price Fetching

Category	Severity	Location	Status
Optimization	● Low	DecreasePositionCollateralUtils.sol	Unresolved

Description

In the `payForCost` function, if the collateral token amounts are insufficient to pay for the cost, the secondary price is fetched to convert the remaining cost into a secondary token amount.

However the secondary token price, which is always the pnl token price, is re-fetched upon every `payForCost` call.

Meanwhile there is an existing `cache.pnlTokenPrice` on the `PositionUtils.DecreasePositionCache memory cache` parameter.

Recommendation

Accept the `pnlTokenPrice` as an argument to the `payForCost` function and pass the `cache.pnlTokenPrice` as the value upon each `payForCost` call.

Resolution

DPCU-7 | initialCollateralDeltaAmount Unexpected Adjustment

Category	Severity	Location	Status
Unexpected Behavior	Low	DecreasePositionCollateralUtils.sol: 485-505	Unresolved

Description

The comment on line 485 notes that “the priceImpactDiffUsd has been deducted from the output amount or the position's collateral”, but in fact the priceImpactDiffUsd can be deducted from the secondaryOutputAmount in a subset of cases.

Additionally, there are several cases in which the priceImpactDiffUsd is deducted from a subset/combination of the three.

In cases where the priceImpactDiffUsd is deducted in any part from either the outputAmount or the secondaryOutputAmount subtracting the full priceImpactDiffAmount from the initialCollateralDeltaAmount does not accurately reflect the portion of the priceImpactDiffUsd that was deducted directly from the collateral. Therefore this adjustment does not make the effect on the collateral amount predictable.

Recommendation

Consider adjusting the initialCollateralDeltaAmount by the exact amount deducted from the collateral (or even just the amount paid in the collateral token to be somewhat accurate, while maintaining simplicity) when paying for the priceImpactDiffUsd on lines 388-428.

Resolution

GLOBAL-4 | Empty Event Data

Category	Severity	Location	Status
Events	Low	Global	Unresolved

Description

Throughout the codebase, there are instances of `EventData` being declared and being returned empty.

For example, `IncreaseOrderUtils.processOrder` returns empty event data. In comparison `DecreaseOrderUtils.processOrder` and `SwapOrderUtils.processOrder` return populated event data.

Recommendation

Ensure that event data is populated where necessary.

Resolution

DPCU-8 | Inaccurate Comment

Category	Severity	Location	Status
Comments	Low	DecreasePositionCollateralUtils.sol: 101	Unresolved

Description

The comment on line 101 states that “then priceImpactUsd would be $20”, however this should refer to the priceImpactDiffUsd instead.

Recommendation

Replace priceImpactUsd with priceImpactDiffUsd in the comment.

Resolution

POSU-2 | getPositionPnlUsd No Longer Needs Separate Prices

Category	Severity	Location	Status
Superfluous Code	Low	PositionUtils.sol: 165	Unresolved

Description

Since the base PnL is calculated, now independent of price impact, the `getPositionPnlUsd` function no longer needs to use a separate index token price to compute the `poolPnl` for capping calculations.

Recommendation

The `executionPrice` can be renamed to the `indexTokenPrice` and this price can be used for all calculations.

Resolution

IPU-2 | Superfluous if Case

Category	Severity	Location	Status
Superfluous Code	Low	IncreasePositionUtils.sol: 139-141	Unresolved

Description

The `if (cache.sizeDeltaInTokens < 0)` case will never be satisfied as the `cache.sizeDeltaInTokens` is a `uint256` value.

Recommendation

Remove this superfluous `if` case.

Resolution

GLOBAL-5 | Users Negatively Impacted By Price Spread

Category	Severity	Location	Status
Documentation	● Low	Global	Unresolved

Description

When users are depositing, the tokens in the `poolAmount` during the `getPoolValueInfo` function are valued at the maximum price, meanwhile the user's deposits are valued at the minimum price.

When users are withdrawing, the tokens in the `poolAmount` during the `getPoolValueInfo` function are valued at the minimum price, meanwhile the `longTokenPoolUsd`, `shortTokenPoolUsd` and `totalPoolUsd` are valued at the maximum price.

Therefore users are negatively impacted by large spreads when depositing and withdrawing from a market.

Additionally, it is noteworthy that this will in some cases leave behind a non-trivial amount of tokens in the `poolAmount` after all depositors have withdrawn.

Recommendation

It should be well documented that depositors and withdrawers are negatively impacted by the price spread.

Resolution

GLOBAL-6 | Missing NatSpec

Category	Severity	Location	Status
Documentation	Low	Global	Unresolved

Description

Several functions throughout the codebase are missing NatSpec documentation:

- `getExecutionPrice`
- `payForCost`
- `handleEarlyReturn`
- `getEmptyFees`

Recommendation

Add the relevant documentation to functions where NatSpec is lacking.

Resolution

MKTU-7 | Stable Funding Factor Liquidation Risk

Category	Severity	Location	Status
Configuration	Low	MarketUtil.sol: 1944-1946	Unresolved

Description

In the event where a stable funding factor is set or unset it may cause positions to unexpectedly become liquidatable due to a stepwise increase/decrease in the factor and the resulting funding fees calculated.

Recommendation

Document that positions may unexpectedly experience an increase/decrease in funding fees due to the modification of the stable funding factor.

Resolution

ARR-1 | Check Odd Gas Optimization

Category	Severity	Location	Status
Optimization	Low	Array.sol: 122	Unresolved

Description

Modulo 2 is used to determine if the length of the `arr` array is odd, however `& 1` is a more efficient alternative.

Recommendation

Modify the check from `arr.length % 2 == 1` to `arr.length & 1 == 1`.

Resolution

GLOBAL-7 | Liquidation Fee

Category	Severity	Location	Status
Suggestion	Low	Global	Unresolved

Description

Currently there is no additional fee that a trader incurs when they are liquidated. The trader is simply forced to settle their fees/losses. This way a liquidation is roughly equivalent to a stop loss.

This may allow traders to open high leverage positions and gain an advantage from the fact that they will simply be “stopped out” when they are liquidatable.

Additionally traders may gain an advantage by shifting the monetary impact of an insolvent close onto the protocol in the event that price gaps significantly and a high leverage position is immediately under water. Traders are not negatively impacted by liquidations so they are more likely to take on these positions and push these losses onto the pool depositors in the case of insolvent closes.

Recommendation

Consider adding a liquidation fee to make liquidations sufficiently unattractive for traders. The fee may be allocated to the pool to offset the potential financial impact of insolvent closes.

The liquidation fee may be configured to 0 in the vast majority of cases. However it may prove useful in the event of high leverage position manipulations.

Resolution

IOU-2 | Swap Before Updating Borrowing State

Category	Severity	Location	Status
Protocol Manipulation	Low	IncreaseOrderUtils.sol: 23	Unresolved

Description

During execution of increase orders, a swap is performed to receive the position's collateral, shifting the pool amounts. This swap is performed before the borrowing factor is updated with `updateFundingAndBorrowingState` which will rely on the balance of backing tokens in the pool to compute the current borrowing fees.

A trader can use the collateral token swap during increase to manipulate the balance of backing tokens and therefore minimize the borrowing fees that are recorded for their position.

Recommendation

Monitor the swap fee to dissuade any borrowing fee manipulation and document such behavior.

Resolution

GLOBAL-8 | Floating Pragma Version

Category	Severity	Location	Status
Best Practices	● Low	Global	Unresolved

Description

Throughout the codebase, the `.sol` files use a floating pragma with ^0.8.0. There is a list of known bugs in many 0.8 Solidity versions that were fixed in subsequent releases: https://github.com/ethereum/solidity/blob/develop/docs/bugs.json

Furthermore, if compiled with 0.8.20 there may be unexpected reverts when deployed as many chains still do not support the `PUSH0` opcode.

Recommendation

Use a static pragma.

Resolution

IPU-3 | Unnecessary perSizeValues Initialization

Category	Severity	Location	Status
Optimization	Low	IncreasePositionUtils.sol: 77-104	Unresolved

Description

During the increase position logic, the funding `perSize` values are stamped on a new position with zero size. However, the entire `if` case and `perSize` initialization is unnecessary as the funding fees and claimable amounts will be calculated by multiplying the position size with the `diffFactor`.

Initially the position size will be zero, resulting in 0 funding fees and claimable amounts, even with a non-zero `diffFactor`. The funding fees `perSize` values will then be set for the position later on (lines 170-172). Therefore the initial stamping of the funding `perSize` values for a brand new position are unnecessary

Recommendation

Remove the `if` case where a new position with 0 `sizeInUsd` has the `perSize` values stamped.

Resolution

SWPU-1 | Misleading priceImpactUsd Emitted

Category	Severity	Location	Status
Events	Low	SwapUtils.sol: 351	Unresolved

Description

At the end of a swap, the `priceImpactUsd` is emitted with the `emitSwapInfo` function.

However the `priceImpactUsd` value may not be accurate to how much price impact was actually applied to the swap, depending on if the impact amount was capped by the size of the swap impact pool or not.

Recommendation

Either modify the value passed to the event or add a parameter for `priceImpactAmount`.

Resolution

POSU-3 | Outdated Price Impact Formula

Category	Severity	Location	Status
Documentation	● Low	PricingUtils.sol: 117	Unresolved

Description

According to the README, the formula for price impact calculations should be:

`(initial imbalance) ^ (price impact exponent) * (price impact factor / 2) - (next imbalance) ^ (price impact exponent) * (price impact factor / 2)`

However with latest changes to `PricingUtils.applyImpactFactor()`, the `/ 2` division was removed from the formula.

Recommendation

Update the formula in the documentation.

Resolution

MKTU-8 | Missing swapPath Validation

Category	Severity	Location	Status
Validation	Low	MarketUtils.sol: 2352	Unresolved

Description

The `validateSwapPath` function does not include validation for homogenous markets, which would cause the swap to fail automatically upon execution.

Similarly, there is no validation for duplicate markets in the provided `swapPath` until the swap is attempted during execution.

Recommendation

In the `validateSwapPath` function, verify that there are not any single token markets or duplicate markets in the `swapPath`.

Resolution

DATA-1 | Superfluous Stack Variable

Category	Severity	Location	Status
Optimization	● Low	DataStore.sol: 97	Unresolved

Description

The current value of the `uintValues[key]` is cached as the `uint256 currValue` stack variable. However, the `currValue` variable is only referenced once on the very next line and so therefore can be replaced with a direct access of the mapping.

Recommendation

Access the mapping directly when calculating `nextUint` rather than storing the `currValue`.

Resolution

ERTR-1 | ExchangeRouter Inefficient Loops

Category	Severity	Location	Status
Optimization	Low	ExchangeRouter.sol	Unresolved

Description

Throughout the `ExchangeRouter` several `claimedAmounts` lists are declared of with a length that is subsequently looped over. However this length is recomputed for both the list declaration as well as the `for` loop. This length can be cached to save gas on both the list declaration and upon each iteration of the `for` loop.

Recommendation

Cache the array length and use this cached value in the `for` loop and array declaration.

Resolution

GLOBAL-9 | Empty Swap Path For Swap Orders

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Users can submit valid swap orders that have an empty `swapPath`.

Recommendation

Validate that the `swapPath` length is nonzero for swap orders upon order creation.

Resolution

ARR-2 | Oracle Block Validation Optimizations

Category	Severity	Location	Status
Optimization	● Low	Array.sol	Unresolved

Description

In the `get`, `areEqualTo`, `areGreaterThan`, `areGreaterThanOrEqualTo`, `areLessThan`, `areLessThanOrEqualTo` functions, the length of the `arr` array is not cached during the `for` loop execution.

Additionally, the increment of `i` can be replaced with an `unchecked` block wrapping `++i`.

Recommendation

Implement the recommended optimizations.

Resolution

GLOBAL-10 | Price Impact Incongruence

Category	Severity	Location	Status
Inconsistency	Low	Global	Unresolved

Description

Price impact for deposits is calculated based on the pool amount imbalance from the deposited amounts before fees are accounted for.

However price impact for swaps is calculated after fees are taken from the swapped in amount.

The known issues in `README.md` mention that `Calculation of price impact values do not account for fees`, however this is directly in contradiction to the price impact calculations for swaps.

Recommendation

Consider standardizing on one or the other for parity across deposits and swaps or document this key difference.

Resolution

GLOBAL-11 | Superfluous Price Impact Logic

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

When computing and applying the price impact amounts in both deposits and swaps, the `else` case handles instances where there is $0 of price impact.

However, the accounting logic and corresponding event emissions are unnecessary when the `priceImpactUsd` is computed to be 0.

The `priceImpactUsd` will often be zero when depositing balanced long and short token amounts, depositing into homogenous markets, or making swaps that imbalance the pool as much as it is balanced.

Recommendation

Exclude cases where the `priceImpactUsd` is 0 from the `else` branch to avoid superfluous logic.

Resolution

GLOBAL-12 | Redundant Price Fetching

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

The `willPositionCollateralBeSufficient` function is invoked in `IncreasePositionUtils` as well as `DecreasePositionUtils` where there is a `cache.collateralTokenPrice` available in both cases.

However the `willPositionCollateralBeSufficient` function redundantly fetches the collateral token price with `getCachedTokenPrice`.

Recommendation

Accept the `collateralTokenPrice` as a parameter for the `willPositionCollateralBeSufficient` function to avoid fetching it again.

Resolution

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits