GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

GMX

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, 0xKato, 0xWeiss

Client Firm GMX

Final Report Date March 15th, 2023

Audit Summary

GMX engaged Guardian to review the security of its decentralized synthetics perpetuals exchange. From the 31st of January to the 15th of March, a team of 4 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

📊 Comprehensive code coverage test suite: https://github.com/GuardianAudits/GMX_2

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics/
Commit(s)	bf46b027a7ef80b00dd4451b0282d2e51bb9a24e

Audit Summary

Delivery Date	March 15th, 2023
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	16	16	0	0	0	0
High	9	9	0	0	0	0
Medium	31	31	0	0	0	0
Low	41	41	0	0	0	0

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ADLU	AdlUtils.sol	85caccecc9630eaf43a4128e74ae469575a1f938
BNK	Bank.sol	814ae68a75e21621c3d22c174eea28c45fb2118f
SBNK	StrictBank.sol	8235d39cfa13186c2b4a4fbadfab029bcd8f91d2
CBKU	CallbackUtils.sol	b57c3a07448c6e5d75207ecedd36d924a4ffc575
DCBK	IDepositCallbackReceiver.sol	aed58b8d02e950c17f1e375d1ff4537e20d4d460
OCBK	IOrderCallbackReceiver.sol	b986dcf7d9deb75f6cbb6e630a3f7d2a27f75374
WCBK	IWithdrawalCallbackReceiver.sol	d85b4c126911ed219a4bb13349a35887e9b6db84
ARBS	ArbSys.sol	0d9703a3477e40ccce9b0b526c0c9f4310034496
CHAIN	Chain.sol	020d318af7d3d4ecba2cdf36669c582923611ae9
DATA	DataStore.sol	1e688ae1d74fce8ed806d1c71c7bf9f4f1fc11e3
KEY	Keys.sol	21f3e86a3b97c0da18ae93ab12967cce408b7d46
DEP	Deposit.sol	f15405bbbafc9a723e2a596193919975b25c35e8
DEPS	DepositStoreUtils.sol	aefb9405cbeeeaa26c31a66d124386b34aad1b12
EMIT	EventEmitter.sol	b12353d5f75258c78f7d7eb14082191d3fb3b1e2
DEPH	DepositHandler.sol	dc04f12b714748dde171acf33c4db5498c06c0a4
ORDH	OrderHandler.sol	52b3afd90d01d9b7827995567bdc40260437dd71
WTDH	WithdrawalHandler.sol	04d892483de2a081a012b3f1c9583137dd8468d5
FTU	FeatureUtils.sol	c13c0754b300f9db33673ddda7d10443f5897d24
ODV	OrderVault.sol	74f991769825ba9fc8b98f3be3a5fefc32be7539
DPV	DepositVault.sol	1d19ad5afc0baec27a608a2f53cbb5b6f48f8f26

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
WDV	WithdrawalVault.sol	5cc2b331b13f735dfebc983b9aec705692e0d2a2
GSU	GasUtils.sol	0532683bdb842a447bfdbf15ba0564f4e06e0a75
GOV	Governable.sol	c5b3c7089b41b94e1d36f4e0492bf758e106547a
LIQU	LiquidationUtils.sol	3181ad9d6d64a3bc5428e87464d103381c53ac1b
MKT	Market.sol	a66a2a9127674ffb23d74d7a252f046f98c2e182
MKTF	MarketFactory.sol	77deb5eb5fe4bae2de7471ae66c4f6cdef5a9a92
MKTS	MarketStoreUtils.sol	d0d6795a715d7cec428fe4333a37da5df650b3e7
MKTT	MarketToken.sol	a55f9a9931d906583050b4f01b74b7adbe54cf1d
MKTU	MarketUtils.sol	972329ca2990356c34fb7a6c5077bdaeb90eade9
NCU	NonceUtils.sol	6ec2082417987d5c4e859adefb9b28efb1ed5c39
PFE	IPriceFeed.sol	431babdd9ab4ee30ae9eba84f469620a3d2951f3
OCL	Oracle.sol	d67693b91b26a84ce8e84fd375970ed07dfe840f
OCLM	OracleModule.sol	63cea7c1c2489e757501b219066b0b75e0245c32
OCLS	OracleStore.sol	1e6a95ac567b91c345c1647a82e62d7fd00617e2
OCLU	OracleUtils.sol	96a8ae6230ceeabe0cc471487d1d2fd8a354511e
DOU	DecreaseOrderUtils.sol	3f171729dc3055f1ae7867949aad477c8941d7d5
IOU	IncreaseOrderUtils.sol	d813868465dde99059b1ee5bffe49f80bcb04462
ORD	Order.sol	090ce71a5e61445b7288267bf157dc4927f4e6a1
BOU	BaseOrderUtils.sol	ceafe9d2c4e14b043022a5c23c37cc30be72bec2
ORDS	OrderStoreUtils.sol	3dfdd3dcaf4b55eab0ac21639c096994062fa266
FEU	FeeUtils.sol	0f7130ee04d5f4bc82c7d5a72d5c83ac226e9918

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ORDU	OrderUtils.sol	0ecba5274f336d863fa7085db8f3b4284a052a19
SWOU	SwapOrderUtils.sol	65051e5535ff27531518d29b779792df02e191c7
DPU	DecreasePositionUtils.sol	78489065744b179300ba784141320452e62e7334
IPU	IncreasePositionUtils.sol	7ad663427c15de3ff78692867999ff10bb90f261
POS	Position.sol	d6dced94def32ea2786c29749a3bea2f4e9e4202
POSU	PositionUtils.sol	8ae17f40ccf9a218fd64b95a3d9200f915ed6a39
PRICE	Price.sol	c1f87807a20c43c1710d1e3c3e628e265cd5686e
PPU	PositionPricingUtils.sol	41c9eb4a6e49f22d376334df5dfa4dc2cfb1e901
PRU	PricingUtils.sol	2f55f33f64f01e06b0aa5b928651ec4496ce4ba9
SPRU	SwapPricingUtils.sol	73a5c8671e031ff38415c3316df676a84ab30524
READ	Reader.sol	12343e67be606e67b69ca4c8da7f9a9d6e24745e
IREFS	IReferralStorage.sol	f61d9bb3c2ec803d3b97c1e7f4faca4f1e517bf6
REFT	ReferralTier.sol	8a34d5e24b6a317b063ebd59d85fa1fec9307ea7
REFU	ReferralUtils.sol	617bf4115a4d5a42f4fff58c37fd5651ad74af0b
REFS	ReferralStorage.sol	086c0102b673a95198c213003ba1e0882dbd6a87
ROLE	Role.sol	86935a3af0c782e711076d1a2ad2222bda7185fa
ROLEM	RoleModule.sol	b3e74811c0f6a46ff26da1474fd48969314d4938
ROLES	RoleStore.sol	5131089f2c42508c33ab3ef7febdc29132cf92b3
DEPU	DepositUtils.sol	c871656056f44bc02dd8dcd7cb18e18f3bbd44aa
EDPU	ExecuteDepositUtils.sol	0db7ed99684c406a0ac926fb04557c47c5e767d3
DPCU	DecreasePositionCollateralUtils.sol	88dad5c90fea4845fb956cac03a1c08dd2bbc99c

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ERTR	ExchangeRouter.sol	fe54c82d3771dd6c8bb2fb5d2bf36e4ec44a2e68
RTR	Router.sol	0fde38bae3c62565cda7fec0ba521a46611d6e32
SWPH	SwapHandler.sol	9e3bb4bb999a70390ff2be5f447a7d4ffd5699c5
SWPU	SwapUtils.sol	f0a75866cc0a8191d1f4fc37d2b32c9fb64b9aa9
TIME	Timelock.sol	e75d66d59c0d7dc531545527aeabfea1515b9167
IWNT	IWNT.sol	972554584395e769df3392828d0e43adc74801f4
TU	TokenUtils.sol	dfbaa478edbc1f862cf0649d7c7f91debb82db1b
ARR	Array.sol	475174aabc82306f52589c927641ce4c85f79e29
BIT	Bits.sol	c7fa3c25af05c172cff6faccef14182665b875ba
CLC	Calc.sol	6ce439db40dd185a189d93b121441d8ee45717cb
ENM	EnumerableValues.sol	36354b53a39c4fb584313f8d3aac8e2b091d90a2
MC	Multicall3.sol	2388e6a306c163da07ff92daeb4f7c8e95828065
PMC	PayableMulticall.sol	d4748b4b4fa4715f63fac17d0f406627d64658da
PREC	Precision.sol	327da594b4f829b1dc21c548bf4e7a3c176aba79
WTD	Withdrawal.sol	86a4ddc39df006b71c0ffc8366f401845488a9d1
WTSU	WithdrawalStoreUtils.sol	bf7b0e22c6c1975dd05f2663e161b67d482f9434
WTDU	WithdrawalUtils.sol	a63f7511edfa427108e6a0bfd9e79a606e929a4a
CBU	CallbackUtils.sol	b57c3a07448c6e5d75207ecedd36d924a4ffc575
CON	Config.sol	07a46298bf488b17ce72620cb9d9a0ea87467930

Audit Scope & Methodology

Vulnerability Classifications

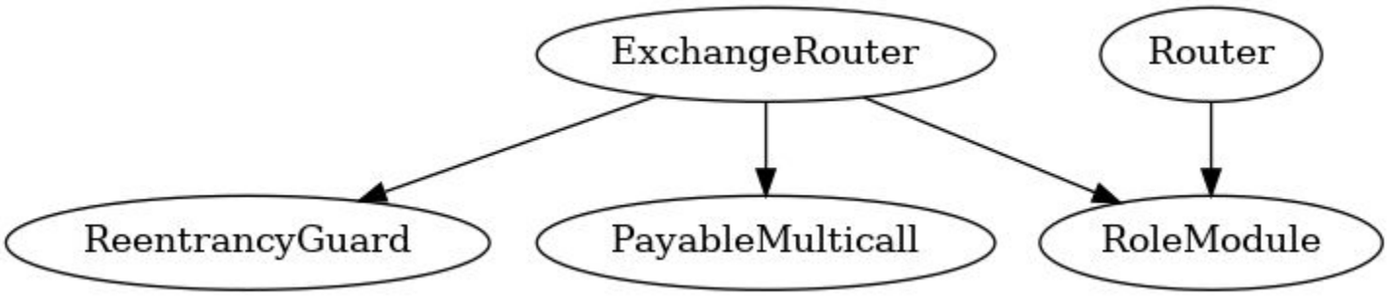
Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract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

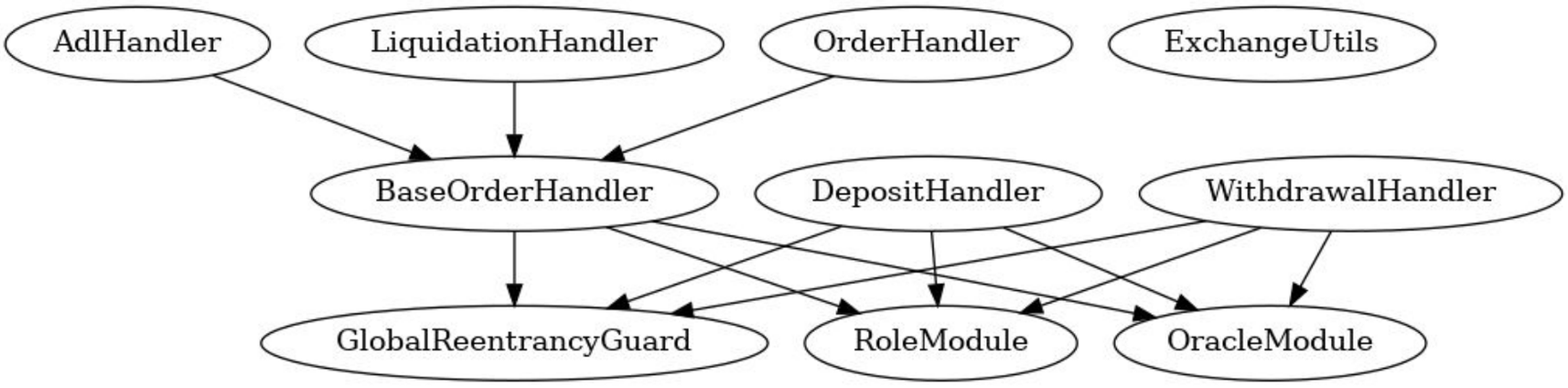
Methodology

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
GLOBAL-1	Homogeneous Markets Double Count Value	Double Counting	Critical	Unresolved
ORDU-1	Unbounded swapPath Length	Gas Manipulation	Critical	Unresolved
PPU-1	Open Interest Value Uninitialized	Logical Error	Critical	Unresolved
MKTU-1	Impact Pool Included In Pool Value	Logical Error	Critical	Unresolved
MKTU-2	Funding Fees Partially Paid	Logical Error	Critical	Unresolved
MKTU-3	Unclaimable Collateral	Logical Error	Critical	Unresolved
GLOBAL-2	Blacklisted Addresses Can Exploit The Exchange	Blacklisted Addresses	Critical	Unresolved
DPCU-1	Wrong Token Amount Applied	Logical Error	Critical	Unresolved
CBU-1	Malicious Revert Bytes	Gas Manipulation	Critical	Unresolved
DPCU-2	LimitDecrease Gamed With EmptyPosition Error	Protocol Manipulation	Critical	Unresolved
MKTU-4	60 Decimals Of Precision Overflow	Overflow	Critical	Unresolved
GSU-1	Missing Swap Gas Estimation	Gas Attack	Critical	Unresolved
IPU-1	Rounding Leads To Risk Free Trade	Protocol Manipulation	Critical	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
CON-1	_validateRange Prevents Critical Values Being Set	Logical Error	Critical	Unresolved
GLOBAL-3	Referral Codes Used To Game Orders	Protocol Manipulation	Critical	Unresolved
GLOBAL-4	positionIncreasedAtBlock Used To Game Orders	Protocol Manipulation	Critical	Unresolved
MKTU-5	Fee Receiver Amount Included In Pool Value	Logical Error	High	Unresolved
MKTU-6	Rounding Error Causes Market Insolvency	Precision Loss	High	Unresolved
PPU-2	Price Impact For Trader Not Equal Price Impact For Pool	Accounting Error	High	Unresolved
DPU-1	Outdated Fees For Liquidation Check	Logical Error	High	Unresolved
ORDH-1	Unaccounted Gas Expenditure When Setting Prices	Gas Attack	High	Unresolved
EDPU-1	Adjusting Long And Short Token Amounts Incorrect	Logical Error	High	Unresolved
MKTU-7	Total Borrowing Fees Outdated	Logical Error	High	Unresolved
POSU-1	Profit Included In Remaining Collateral	Logical Error	High	Unresolved
GLOBAL-5	Reference Exchange Manipulation	Protocol Manipulation	High	Unresolved
ORDH-2	Frozen Orders Can Not Be Simulated	Logical Error	Medium	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
TU-1	Call Return Value Gas Manipulation	Gas Manipulation	Medium	Unresolved
ORDH-3	Short Term Risk Free Trade With Limit Orders	Protocol Manipulation	Medium	Unresolved
ADLU-1	Direct Use Of block.number	Logical Error	Medium	Unresolved
EDPU-2	No Pool Amount Validation For Positive Impact	Logical Error	Medium	Unresolved
GLOBAL-6	Block Stuffing Attack	Gas Manipulation	Medium	Unresolved
SWPU-1	Lack Of Validation For Homogenous Markets	Logical Error	Medium	Unresolved
GSU-2	Gas Price Deficit	Gas Prices	Medium	Unresolved
DPCU-3	swapProfitToCollateralToken Invalid Impact	Accounting Error	Medium	Unresolved
WTDU-1	Users Can Game Withdrawals	Protocol Manipulation	Medium	Unresolved
DPCU-4	indexToken vs. pnlToken Arbitrage	Arbitrage Opportunity	Medium	Unresolved
DPU-2	Drain Keeper's Gas Through Liquidations	Gas Attack	Medium	Unresolved
PPU-3	No Lower Bound On Virtual Inventory Price Impact	Logical Error	Medium	Unresolved
PPU-4	borrowingFeeAmountForFeeReceiver Double Counted	Double Counting	Medium	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
MKTU-8	Precision Loss For Funding Fees	Precision Loss	Medium	Unresolved
SPRU-1	Double Counting Swap Imbalance	Double Counting	Medium	Unresolved
DPU-3	Position Unexpectedly Closed	Unexpected Behavior	Medium	Unresolved
GLOBAL-7	Funding Fees Accumulate In Disabled Markets	Unexpected Behavior	Medium	Unresolved
ADLU-2	Latest ADL Block Updated Without ADL State Change	Logical Error	Medium	Unresolved
ORDH-4	Risk Free Trade With Disabled Feature	Protocol Manipulation	Medium	Unresolved
DPCU-5	Remaining Collateral Adjusted To Revert	Logical Error	Medium	Unresolved
WTDU-2	Swap Required To Specify Output Amount	Slippage	Medium	Unresolved
GLOBAL-8	Lack Of Slippage Protection When Swapping	Slippage	Medium	Unresolved
GSU-3	No Way For Users To Claim Excess Execution Fee	Lost Funds	Medium	Unresolved
ADLU-3	Two Separate ADL Factors	Logical Error	Medium	Unresolved
SWPU-2	Swaps Prevented When They Improve The Pool	Logical Error	Medium	Unresolved
ORDH-5	Gas Used Is Overestimated	Logical Error	Medium	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-9	Pool State Leading To Withdrawals Being Bricked	Trapped Funds	Medium	Unresolved
EDPU-3	Market Token Price Below Allowed Amount	Logical Error	Medium	Unresolved
CLC-1	Bounded Sub Can Underflow	Logical Error	Medium	Unresolved
GLOBAL-10	Double Fee May Make A Position Liquidatable	Double Counting	Medium	Unresolved
PPU-5	Borrowing Fees Maximized Twice	Logical Error	Low	Unresolved
POSU-2	Unexpected Closing Of Positions	Logical Error	Low	Unresolved
POSU-3	Swapped Parameters	Typo	Low	Unresolved
WTDU-3	Overflow Risk	Overflow	Low	Unresolved
DPU-4	Affiliate Rewards Upon Liquidation	Incentives	Low	Unresolved
DPU-5	Worst Case Estimation Unused	Validation	Low	Unresolved
POSU-4	Funding Fees Sent To Receiver	Unexpected Behavior	Low	Unresolved
PPU-6	Sandwich Attack For Price Impact	Protocol Manipulation	Low	Unresolved
IOU-1	Limit Increase With Swap Path May Be Griefed	Protocol Manipulation	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
SWPU-3	Event Getting Incorrect Value	Logical Error	Low	Unresolved
DPCU-6	Lost Funding Fees	Lost Funds	Low	Unresolved
TIME-1	Bespoke Key Used	Logical Error	Low	Unresolved
CLC-2	Roundup Division Rounds Down Instead Of Up	Documentation	Low	Unresolved
ORDH-6	Limit Order Cancellation Logic	Protocol Manipulation	Low	Unresolved
OCLU-1	Misnamed Variable	Readability	Low	Unresolved
GLOBAL-11	Additional Feature Controls	Controls	Low	Unresolved
SWPU-4	Missing Check For tokenIn	Validation	Low	Unresolved
OCL-1	Unsorted Max Oracle Block Numbers	Validation	Low	Unresolved
PREC-1	SafeCast Revert	Documentation	Low	Unresolved
BOU-1	Set Exact Order Price Using Older Price	Logical Error	Low	Unresolved
BNK-1	Future Proof Receive Function	Future Proofing	Low	Unresolved
DPU-6	Inaccurate Fees May Be Emitted	Events	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
OCL-2	Missing Check	Validation	Low	Unresolved
DPU-7	Collateral May Not Be Sufficient	Logical Error	Low	Unresolved
MKTU-9	Duplicated Code	Optimization	Low	Unresolved
PPU-7	Variable Reuse	Optimization	Low	Unresolved
POSU-5	Use Cheaper Branch	Optimization	Low	Unresolved
GLOBAL-12	Internal Library Functions	Visibility Modifiers	Low	Unresolved
DPCU-7	Use Cached Variable	Optimization	Low	Unresolved
POSU-6	Function Reuse	Optimization	Low	Unresolved
GLOBAL-13	Initialization Of Default Values	Optimization	Low	Unresolved
GLOBAL-14	Initialization Of Default Values	Optimization	Low	Unresolved
GLOBAL-15	Cached Array Length	Optimization	Low	Unresolved
GLOBAL-16	Cached Array Length	Optimization	Low	Unresolved
GLOBAL-17	Greater Than Zero Check	Optimization	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
SWPU-5	Outdated NatSpec	Documentation	Low	Unresolved
BOU-2	Outdated NatSpec	Documentation	Low	Unresolved
MKTU-10	Outdated NatSpec	Documentation	Low	Unresolved
POSU-7	Outdated NatSpec	Documentation	Low	Unresolved
BOU-3	Outdated NatSpec	Documentation	Low	Unresolved
POSU-8	Outdated NatSpec	Documentation	Low	Unresolved

GLOBAL-1 | Homogeneous Markets Double Count Value

Category	Severity	Location	Status
Double Counting	● Critical	Global	Unresolved

Description PoC

When using the `getPoolValue` function for markets with identical long and short backing tokens, the `cache.longTokenAmount` and `cache.shortTokenAmount` represent the same token amount.

However both of these token amounts are counted in the value of the pool. Therefore the deposited pool value is doubled.

Additionally, several validations e.g. `validateReserve`, `validateMaxPnl`, and `validatePoolAmount` among others are immediately invalidated as they errantly count the same exact token amount for both the short and long side. This way the single backing token pool is effectively double counted as backing both long positions and short positions.

Similarly open interest in these markets is double counted in the `getNextFundingAmountPerSize` function, producing a completely invalid calculation for funding fees.

Recommendation

Reconsider if markets with the same backing `longToken` and `shortToken` should be possible. If they should, then handle their accounting/validation separately.

ORDU-1 | Unbounded swapPath Length

Category	Severity	Location	Status
Gas Manipulation	● Critical	OrderUtils.sol: 49	Unresolved

Description PoC

When creating an order there is no validation that the `swapPath` is under a certain max length. This allows malicious users to create risk-free trades on the exchange.

Notably, among other ways, a trader may submit a `MarketIncrease` order with a `swapPath` that puts the order just over the block gas limit when combined with a callback that consumes nearly the entire `maxCallbackGasLimit`. When a trader wishes the trade to be executed using outdated prices, they can toggle the callback contract to only consume a very small amount of gas, enabling the order to be executed and recorded in a block.

Recommendation

Add validation on the max length for the `swapPath` of orders to protect the exchange from the entire class of `swapPath` gas manipulation attacks.

PPU-1 | Open Interest Value Uninitialized

Category	Severity	Location	Status
Logical Error	● Critical	PositionPricingUtils.sol: 315-316	Unresolved

Description PoC

In the `getNextOpenInterestParams` function the `nextLongOpenInterest` and `nextShortOpenInterest` variables are not initialized from the default values and only one is set in either of the `params.isLong` cases.

This drastically misrepresents the open interest balance of the market and yields nonsensical price impact calculations.

Recommendation

Initialize each of the `nextLongOpenInterest` and `nextShortOpenInterest` values to the current open interest on each side.

MKTU-1 | Impact Pool Included In Pool Value

Category	Severity	Location	Status
Logical Error	● Critical	MarketUtils.sol: 317-318	Unresolved

Description PoC

In the `getPoolValue` function, the position impact pool value is added to the value of the pool. However the position impact pool is comprised of a notional value of synthetic index tokens, therefore it should not be added to the pool value.

When a trader is negatively impacted, two things happen:

1. `indexTokens` are taken from the trader and allocated to the position impact pool.
2. The trader immediately experiences a loss on their PnL.

This results in the pool accounting for this negative impact amount twice. Once for the `indexTokens` that are allocated to the position impact pool. And a second time for the negative PnL that the trader has just experienced.

This double counting invalidates the market's accounting system and does not allow depositors to withdraw all of their `MarketTokens`.

Recommendation

Do not account the position impact pool as a part of the net `poolValue`.

MKTU-2 | Funding Fees Partially Paid

Category	Severity	Location	Status
Logical Error	● Critical	MarketUtils.sol: 906-909	Unresolved

Description PoC

In the `getNextFundingAmountPerSize` function, the `fundingAmountPerSizePortion` values are calculated by dividing the corresponding `fundingUsd` by the open interest of both longs and shorts. This is done to get a value that is to be received/paid for each collateral type for each position direction.

This logic makes sense in the case where users are being paid the funding fees since they are able to claim both the long and short tokens. However users are only able to pay the `fundingAmountPerSizePortion` value for their respective collateral token.

This errantly reduces the amount the `fundingAmountPerSizePortion` that is paid for each corresponding collateral token as the amounts are divided by the entire open interest of the side that is paying funding fees. As a result, only a portion of the funding fees are being paid while the entire amount can be claimed, and a deficit in the pool accounting is created which will force markets into insolvency over time.

Recommendation

Implement separate logic for the side that is paying the funding fees where the funding fees are not spread across the entirety of the open interest for that side, but rather the open interest of that side that is able to pay the particular token through their collateral.

E.g.

`cache.fps.fundingAmountPerSizePortion_LongCollateral_LongPosition = getPerSizeValue(cache.fundingUsdForLongCollateral / prices.longTokenPrice.max, cache.oi.longOpenInterestWithLongCollateral);`

MKTU-3 | Unclaimable Collateral

Category	Severity	Location	Status
Logical Error	● Critical	MarketUtils.sol: 575	Unresolved

Description PoC

When users attempt to claim their collateral, the `adjustedClaimableAmount` is asserted to be `<` the `claimedAmount`, otherwise the tx reverts. However if a user has not claimed any of their collateral the claimed amount will be 0 and therefore the `adjustedClaimableAmount` cannot be strictly less than the `claimedAmount`.

Therefore users are unable to claim their collateral.

Recommendation

Modify the `if` statement to revert in the case where `adjustedClaimableAmount < claimedAmount`.

GLOBAL-2 | Blacklisted Addresses Can Exploit The Exchange

Category	Severity	Location	Status
Blacklisted Addresses	● Critical	Global	Unresolved

Description PoC

Addresses that are blacklisted for popular ERC20 tokens such as USDC can be leveraged to exploit the exchange in a number of ways.

These addresses cannot be liquidated in any case where they would be transferred back a leftover collateral amount in a token which they are blacklisted for.

Among other ways, blacklisted addresses can execute risk-free trades using `MarketIncrease` orders in the following way:

1. Force the collateral swap to fail via low liquidity in a niche market.
2. The order cannot be cancelled since the cancellation would attempt to send the token that the user is blacklisted for.
3. Therefore the order will remain in the `dataStore` until the liquidity is added.
4. Deposit liquidity into the low liquidity market so the `MarketIncrease` can go through when the attacker wants it to, using out of date prices for a risk-free trade.

Recommendation

Be extremely cautious when adding markets with tokens that include a blacklist. Consider implementing checks to see if users are blacklisted and denying them service to the relevant markets.

DPCU-1 | Wrong Token Amount Applied

Category	Severity	Location	Status
Logical Error	● Critical	DecreasePositionCollateralUtils.sol: 347	Unresolved

Description PoC

In the case where the `remainingCollateral` for a position is negative during a liquidation, the `getLiquidationValues` function is called. The `values.pnlTokenForPool` is used in the returned values. However the `values.pnlAmountForPool` computed on line 347 is strictly a collateral token amount.

A position may be in profit and still be liquidated due to the `minCollateralUsdForLeverage` combined with a depreciation in the user's collateral token price.

Therefore the `pnlTokenForPool` may be different from the collateral token, in which case a `shortToken` amount could be applied as a `longToken` amount or vice-versa. This will drastically perturb the `poolAmount` for the `pnlTokenForPool` in such a way that could leave the market insolvent or simply cause extreme loss for the market depositors.

Recommendation

Adjust the `getLiquidationValues` function so that it accounts for the cases where traders are being liquidated in profit.

CBU-1 | Malicious Revert Bytes

Category	Severity	Location	Status
Gas Manipulation	● Critical	CallbackUtils.sol	Unresolved

Description PoC

In each callback function the `bytes memory reasonBytes` returned from the third party callback contract in the event of an error is loaded into memory in the `catch` case. `reasonBytes` can be potentially very large and therefore extremely gas intensive when it is copied into memory.

Furthermore, the catch block continues to perform computation with `reasonBytes`, first parsing the error message with `ErrorUtils.getRevertMessage` and also emitting an event that contains `reasonBytes`.

An attacker may simply implement a callback contract that reverts with an extremely large `reasonBytes` so that the execution tx would require more gas than the block gas limit. The attacker can then toggle the callback contract to no longer revert when they want their order to be executed successfully, enabling a risk-free trade.

In another attack, a trader could observe the keeper's execution tx in the mempool and front-run it to toggle the callback contract to revert with a large `reasonBytes`. This would cause the keeper's execution tx to consume an unforeseen amount of gas and likely run out of gas and fail. The attacker could leverage this in a similar manner to create a risk-free trade opportunity.

Recommendation

Do not load the third party callback contract's error `reasonBytes` into memory.

DPCU-2 | LimitDecrease Gamed With EmptyPosition Error

Category	Severity	Location	Status
Protocol Manipulation	● Critical	DecreasePositionCollateralUtils.sol: 219	Unresolved

Description PoC Video

Removing all of the collateral from a position will result in the `EmptyPosition` error, which will be retried for `LimitDecrease` orders.

An attacker can leverage this by creating a `LimitDecrease` order that initially only reduces their `position.sizeInUsd` by half, but reduces their collateral to 0. The `LimitDecrease` will continue to result in the `EmptyPosition` error until the attacker creates a `MarketDecrease` order that reduces the size of their position by half. Now when the original `LimitDecrease` is executed, it will close the position and no longer revert with the `EmptyPosition` error.

A malicious trader can leverage this to make a risk-free trade with their `LimitDecrease`.

Recommendation

Revert with the `InsufficientCollateral` error, which is not retried, in the case where `values.remainingCollateralAmount` is 0.

MKTU-4 | 60 Decimals Of Precision Causes Overflow

Category	Severity	Location	Status
Overflow	● Critical	MarketUtils.sol: 1657, 1737	Unresolved

Description PoC

In the `getPoolValue` function, the `cache.totalBorrowingFees` utilizes 60 decimals of precision, therefore `Precision.applyFactor(cache.totalBorrowingFees, cache.borrowingFeeReceiverFactor)` will have 60 decimals of precision.

Whenever an LP wants to withdraw after borrowing fees have been accumulated, the call to `marketTokenAmountToUsd` in `WithdrawalUtils.sol` would overflow since the pool value is multiplied by the market token amount which has 18 decimals of precision.

Furthermore, the amount of market tokens depositors receive will be drastically reduced as the pool value is inflated.

Recommendation

Do not use 60 decimals of precision for the `cache.totalBorrowingFees` or account for this additional precision when applying the factor.

GSU-1 | Missing Swap Gas Estimation

Category	Severity	Location	Status
Gas Attack	● Critical	GasUtils.sol: 150	Unresolved

Description PoC

The `estimateExecuteWithdrawalGasLimit` does not account for either the `longTokenSwapPath` or the `shortTokenSwapPath`, therefore the keeper will not be remunerated for users using the swap feature on withdrawals.

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

Recommendation

Add gas estimation logic for the `longTokenSwapPath` and `shortTokenSwapPath` in the `estimateExecuteWithdrawalGasLimit` function, similar to the estimation for deposits.

IPU-1 | Rounding Leads To Risk Free Trade

Category	Severity	Location	Status
Protocol Manipulation	● Critical	IncreasePositionUtils.sol: 110	Unresolved

Description PoC Video

In `increasePosition`, the `sizeDeltaInTokens` is rounded down for long positions. An attacker can provide a `sizeDeltaUsd` that is 1 wei less than their `triggerPrice` and have their `sizeDeltaInTokens` rounded to 0. In this case a `LimitIncrease` would revert with the `EmptyPosition` error.

The attacker can make a `LimitIncrease` long for a niche market with low open interest where it is easy to manipulate the price impact by controlling the open interest. The attacker can then manipulate the open interest such that their `LimitIncrease` long is positively price impacted, meaning their `executionPrice` is decreased. A reduction in the `executionPrice` would cause the `sizeDeltaInTokens` to no longer be rounded to 0, and the order to no longer revert with an `EmptyPosition` error.

The attacker can leverage this with a large `initialCollateralDeltaAmount` and a `swapPath` that allows them to take advantage of outdated prices for a risk-free trade.

Recommendation

Do not allow users to create position orders with any `sizeDeltaUsd` less than a particular value such as 1e30 e.g. $1.

CON-1 | _validateRange Prevents Critical Values Being Set

Category	Severity	Location	Status
Logical Error	● Critical	Config.sol: 244	Unresolved

Description PoC

The `_validateRange` function does not perform any validation on the value, but rather reverts for specific keys such as the `Keys.SWAP_FEE_FACTOR` and `Keys.POSITION_FEE_FACTOR`.

This restricts the protocol from setting these crucial values for the exchange.

Recommendation

Update the `_validateRange` function to perform validation on the value being passed rather than simply reverting for certain keys.

GLOBAL-3 | Referral Codes Used To Game Orders

Category	Severity	Location	Status
Protocol Manipulation	● Critical	Global	Unresolved

Description PoC Video

A trader is able to update their `referralCode` at any time by creating a new order through the `ExchangeRouter`.

A malicious trader can submit a `LimitIncrease` where the `fees.totalNetCostAmount` will be their exact `initialCollateralDeltaAmount`, yielding an `EmptyPosition` error on execution.

The malicious trader can then allow the `LimitIncrease` to be executed once they see that prices have moved in their favor by updating their `referralCode` so that their `fees.totalNetCostAmount` is discounted and now the order will no longer revert with the `EmptyPosition` error.

This attack can also be executed if the `tier` or `traderDiscountFactor` is updated on a trader's current `referralCode` as well.

Recommendation

Do not allow a trader's `totalRebateFactor` or `traderDiscountFactor` to be updated in any way for existing orders.

GLOBAL-4 | positionIncreasedAtBlock Used To Game Orders

Category	Severity	Location	Status
Protocol Manipulation	● Critical	Global	Unresolved

Description PoC Video

`LimitIncrease`, `LimitDecrease`, and `StopLossDecrease` orders depend on the `positionIncreasedAtBlock` for the price ranges that they can be executed with.

A malicious trader may take advantage of past prices by closing their position to reset the `positionIncreasedAtBlock` to 0.

Consider the following attack:

1. A trader makes two `LimitIncrease` orders with the same `triggerPrice`.
2. The first one is successfully executed and the `positionIncreasedAtBlock` is set to 100.
3. The second one is not executed since the valid descending price range is from before the `positionIncreasedAtBlock`.
4. The trader waits and observes that price moves in their favor, and then closes their position with a `MarketDecrease` order. Therefore resetting the `positionIncreasedAtBlock` to 0.
5. Now the second `LimitIncrease` is able to be executed with the out of date price range, enabling a risk-free trade.

Recommendation

Track when the position was closed and add this block number to the validation for `LimitIncrease`, `LimitDecrease`, and `StopLossDecrease` orders so that they cannot be used to abuse past prices.

MKTU-5 | Fee Receiver Amount Included in Pool Value

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol: 315	Unresolved

Description PoC

The pool value is incremented by `Precision.applyFactor(cache.totalBorrowingFees, cache.borrowingFeeReceiverFactor)` which is the portion of borrowing fees going to the `feeReceiver`.

Because this amount is paid to the `feeReceiver` rather than the pool, it should not be counted as part of the pool value. This misrepresents the pool's accounting and incorrectly values the pool for deposits and withdrawals.

Recommendation

Include the portion of pending borrowing fees which will go into the pool rather than the portion allocated for the `feeReceiver`.

MKTU-6 | Rounding Error Causes Market Insolvency

Category	Severity	Location	Status
Precision Loss	High	MarketUtils.sol: 994	Unresolved

Description PoC

When the per-size values are computed, for tokens such as USDC with only 6 decimals of precision and large open interest values, there can be rounding error due to the division that occurs in `toFactor`.

For example:

1. $10,000,000 of open interest will have a total number of 38 digits.
2. 1 USDC will have a total number of 37 digits after multiplying by `FLOAT_PRECISION`.
3. Therefore when calculating the per-size values, precision loss on the order of <10 USDC can occur due to truncation.

Additionally, when calculating the per-size values, the resulting values are often not exact multiples of each other.

For example:

1. Open Interest is $10,000 short and $20,000 long both with the long token as collateral.
2. Assume `cache.fundingUsdForShortCollateral / prices.shortTokenPrice.max` yields `57600399999999999`.
3. `57600399999999999 / 10,000 = 5760039999999` will be the short funding factor magnitude.
4. `57600399999999999 / 20,000 = 2880019999999` will be the long funding factor.
5. However `2880019999999 / 5760039999999 = ~0.49999999999991 != 0.5`.

This can result in less funding fees being paid than collected. Such a deficit will eat into the balance of the market and potentially prevent LPers from withdrawing their entire deposits or prevent traders from claiming their funding fees.

Recommendation

Avoid this precision loss for low precision tokens so that the same amount of funding fees are paid as collected. Address the precision lost when the resulting per-size values are not exact multiples.

PPU-2 | Price Impact For Trader != Price Impact For Pool

Category	Severity	Location	Status
Accounting Error	High	PositionPricingUtils.sol: 182	Unresolved

Description PoC

As shown in the following calculations, the price impact experienced by a trader differs from the amount taken out of/into the impact pool which over time will lead to meaningful accounting inconsistencies in the pool.

$$Key: ep = execution\ price,\ lp = latest\ price,\ pia = price\ impact\ amount,\ sd = size\ delta$$

$$Assume\ an\ increase\ order\ with\ positive\ price\ impact.\ Below\ is\ the\ price\ impact\ experienced\ by\ the\ trader.$$

$$pia_1 = \frac{sd}{ep} - \frac{sd}{lp} = \frac{sd * (lp - ep)}{ep * lp}$$

$$Below\ is\ the\ price\ impact\ calculated\ in\ getPriceImpactAmount$$

$$pia_2 = \frac{sd * (lp - ep)}{lp * lp}$$

$$The\ largest\ difference\ arises\ when\ lp\text{-}ep\ is\ maximized$$

Recommendation

Consider using `int256 priceImpactUsd = size.toInt256() * priceDiff / executionPrice.toInt256()` instead of `int256 priceImpactUsd = size.toInt256() * priceDiff / _latestPrice.toInt256()` on line 182.

DPU-1 | Outdated Fees For Liquidation Check

Category	Severity	Location	Status
Logical Error	High	DecreasePositionUtils.sol: 162	Unresolved

Description PoC

The `isPositionLiquidatable` validation is checked before `PositionUtils.updateFundingAndBorrowingState(params, cache.prices)` is performed.

Therefore the borrowing/funding fees are potentially significantly outdated when being accounted for.

This leads to the liquidation keeper not being able to liquidate positions that would be liquidateable when accounting for borrowing/funding fees and ultimately exposes the market to bad debt once the fees are updated.

Recommendation

Update the funding and borrowing state before checking whether the position is liquidatable.

ORDH-1 | Unaccounted Gas Expenditure When Setting Prices

Category	Severity	Location	Status
Gas Attack	High	OrderHandler.sol: 172	Unresolved

Description PoC

The `startingGas` variable is declared inside of the `executeOrder` function. As a result, it will be the amount of gas left after the setting of prices, which is particularly gas intensive.

When calculating how much gas was used in `GasUtils.payExecutionFee`, keepers won't be remunerated for this expenditure which will run a significant deficit over time.

Recommendation

Refactor the way gas expenditure is tracked so that the gas used for the `withOraclePrices` modifier is included in the keeper's remuneration.

EDPU-1 | Adjusting Long and Short Token Amounts Incorrect

Category	Severity	Location	Status
Logical Error	High	ExecuteDepositUtils.sol: 375	Unresolved

Description PoC

In several cases the `getAdjustedLongAndShortTokenAmounts` function reverts or returns a nonsensical result:

1) `poolLongTokenAmount` and `poolShortTokenAmount` access the pool amount for the same token as `longToken` and `shortToken` are the same. As a result, the pool amounts are equal and the `else` case will always be entered.
2) On line 387, when `uint256 diff = poolLongTokenAmount - poolShortTokenAmount` is performed, the larger value is always subtracted from the smaller value causing underflow.
3) On line 396, when `uint256 diff = poolShortTokenAmount - poolLongTokenAmount` is performed, the larger or equal value is being subtracted from the smaller value which will underflow in most cases.
4) On line 400, `adjustedLongTokenAmount - longTokenAmount - adjustedShortTokenAmount` does not set the `adjustedLongTokenAmount` but rather computes the result of subtraction.

Recommendation

Refactor the `getAdjustedLongAndShortTokenAmounts` function to address the above problems.

MKTU-7 | Total Borrowing Fees Outdated

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol: 1739	Unresolved

Description PoC

The `getTotalBorrowingFees` function, which returns the pending borrowing fees, is out of date as it uses an outdated `cumulativeBorrowingFactor` rather than getting the updated factor with `getNextCumulativeBorrowingFactor`.

As a result, LPs who withdraw will have their market tokens worth less than they should be as the pending fees aren't reflected in the pool value.

Additionally, this introduces the opportunity for arbitrages that take advantage of the stepwise increase in `borrowingFees` by forcing an update with a trivial `MarketIncrease` order.

Recommendation

Use `getNextCumulativeBorrowingFactor` instead of `getCumulativeBorrowingFactor` to get the latest pending borrowing fees.

POSU-1 | Profit Included In Remaining Collateral

Category	Severity	Location	Status
Logical Error	High	PositionUtils.sol: 364, 412	Unresolved

Description PoC

The PnL of a position is added to the current collateral when calculating the `remainingCollateralUsd`. This is logically sound when a trader's PnL is negative, as their losses will be subtracted from their collateral upon closing their position.

However, when a position is in profit, there is no effect on the position's collateral since the profits come from the pool. In the case where positions are in profit, adding the profit to the `remainingCollateralUsd` misrepresents how much collateral value actually remains.

Furthermore, if a position is profitable, this profit can be used as collateral to continue to increase the position size. This allows trader's to open positions with far greater leverage than the `minCollateralFactor`.

Recommendation

Do not count profit as a part of the remaining collateral of a position.

GLOBAL-5 | Reference Exchange Manipulation

Category	Severity	Location	Status
Protocol Manipulation	High	Global	Unresolved

Description

An attacker with enough size can manipulate the reference exchanges to influence the median price and take advantage of the price movements.

Recommendation

Carefully monitor the protocol and adjust parameters such as OI caps accordingly. Furthermore, use enough reference exchanges so the median is less likely to be affected by price outliers.

ORDH-2 | Frozen Orders Cannot Be Simulated

Category	Severity	Location	Status
Logical Error	Medium	OrderHandler.sol: 153	Unresolved

Description PoC

In the event that an order is frozen it can no longer be simulated to check for validity since the `msg.sender` will not be a frozen order keeper.

Recommendation

Allow the simulation to bypass the `_validateFrozenOrderKeeper` authentication check.

TU-1 | Call Return Value Gas Manipulation

Category	Severity	Location	Status
Gas Manipulation	Medium	TokenUtils.sol	Unresolved

Description

In the `withdrawAndSendNativeToken` function, `.call` is used to send ether to the `receiver` address. Although `bytes memory data` is commented out, it will still be loaded into memory.

A malicious `receiver` may load unexpectedly large return data into memory and potentially cause the keeper to expend more gas than expected.

The size of the returned data that the `receiver` is able to generate is however constricted by the `gasLimit`, but this form of manipulation may still pose a risk to the system.

Recommendation

Utilize a low level call to avoid loading the returned data into memory:

```
assembly {
    success := call(gasLimit, receiver, amount, 0, 0, 0, 0)
}
```

ORDH-3 | Short Term Risk Free Trade With Limit Orders

Category	Severity	Location	Status
Protocol Manipulation	Medium	OrderHandler.sol	Unresolved

Description PoC

A malicious trader may be able to execute a profitable short-term risk-free trade by creating a limit order, observing the price it will be executed at and optionally front-running the execution to update or cancel the order.

This way the order is cancelled/frozen if price doesn't move in the a direction that benefits the trader in the blocks between where the execution price is from and the current execution block.

Recommendation

Ensure order fees are sufficient to invalidate short term risk-free trades. Otherwise, do not allow users to decide whether or not their order is executed by cancelling or updating the order right before execution.

ADLU-1 | Direct Use Of block.number

Category	Severity	Location	Status
Logical Error	Medium	AdlUtils.sol: 114	Unresolved

Description

`block.number` is used to `setLatestAdlBlock` rather than `Chain.currentBlockNumber()`. This could yield unexpected behavior as the `arbSys.arbBlockNumber()` may differ from the `block.number`.

Recommendation

Utilize the `Chain.currentBlockNumber()` when setting the latest ADL block.

EDPU-2 | No Pool Amount Validation For Positive Impact

Category	Severity	Location	Status
Logical Error	Medium	ExecuteDepositUtils.sol: 367	Unresolved

Description

The `validatePoolAmount` validation check can be circumvented for the `_params.tokenOut` if a user receives a `positiveImpactAmount` which comes in the form of `tokenOut` that pushes the pool amount for `tokenOut` over the `maxPoolAmount`.

Recommendation

Validate that the `_params.tokenOut` pool amount is still within the valid range by using `validatePoolAmount` in the case where the user is positively impacted.

GLOBAL-6 | Block Stuffing Attack

Category	Severity	Location	Status
Gas Manipulation	Medium	Global	Unresolved

Description

Order execution takes ~2,000,000 gas units for a vanilla execution without a `swapPath` or `callbackContract`. On certain chains, such as Avalanche C-chain, the block gas limit can be close to how much gas it takes for order executions. This makes the protocol susceptible to a block stuffing attack.

For example, this transaction on the Avalanche Fuji testnet took over 4,000,000 gas units which is greater than 50% of the gas limit of 8,000,000. An attacker may choose to stuff blocks to delay the execution of their order until the current price moves favorably.

Recommendation

Consider gas optimization strategies alongside measures to remove stale execution tx's from the mempool to prevent such manipulations.

SWPU-1 | Lack Of Validation For Homogenous Markets

Category	Severity	Location	Status
Logical Error	Medium	SwapUtils.sol: 280	Unresolved

Description

For markets where the `shortToken` is the same as the `longToken`, the `validateReserve` function call will only validate reserves for longs. This is because `cache.tokenOut == _params.market.longToken` will always be `true`.

Recommendation

For markets where the `shortToken` is the same as the `longToken`, be sure to validate the reserves for both longs and shorts.

GSU-2 | Gas Price Deficit

Category	Severity	Location	Status
Gas Prices	Medium	GasUtils.sol: 82	Unresolved

Description

In the `validateExecutionFee` function, the current `tx.gasprice` is used to estimate the gas price in the block of the execution. However the current `tx.gasprice` can be significantly different from the `tx.gasprice` actually experienced in the block of execution.

This allows the keeper to expend more gas than the `executionFee` in the case where the `tx.gasprice` is greater in the block of execution.

Recommendation

Be wary of the potential `gasprice` difference and set the `ESTIMATED_GAS_FEE_MULTIPLIER_FACTOR` accordingly.

DPCU-3 | swapProfitToCollateralToken Invalid Impact

Category	Severity	Location	Status
Accounting Error	Medium	DecreasePositionCollateralUtils.sol: 144	Unresolved

Description

When the `swapProfitToCollateralToken` swap is performed, the `pnlAmountForPool` has not yet been decremented from the `poolAmount`. Therefore the price impact calculation as well as subsequent validation checks during `swapProfitToCollateralToken` are based on the `pnlAmountForPool` not being withdrawn from the `poolAmount`, but yet still being swapped.

This leads to inaccurate price impact being applied during the swap as well as validation that is hinged upon a temporary invalid state of the pool accounting.

Recommendation

In the case where the `swapProfitToCollateralToken` swap is performed, account for the user's profit tokens first being removed from the pool before they are used to swap in the pool.

WTDU-1 | Users Can Game Withdrawals

Category	Severity	Location	Status
Protocol Manipulation	Medium	WithdrawalUtils.sol: 197	Unresolved

Description

A malicious user can front-run the execution of their withdrawal and send their market tokens to another address so that the withdrawal execution fails.

The attacker can observe if price moved in their favor between the block where the prices are provided from and the block where their withdrawal execution is happening and decide if they would like their withdrawal to succeed or fail.

An attacker can leverage this using the swaps at the end of a withdrawal to capitalize on outdated prices for any assets in the `longTokenSwapPath` or `shortTokenSwapPath`.

Recommendation

Consider transferring the user's market tokens to a `WithdrawalVault` upon the withdrawal creation, similar to deposits and orders. Otherwise ensure that the withdrawal fees invalidate any possible risk-free trade that could be made.

DPCU-4 | indexToken vs pnlToken Arbitrage

Category	Severity	Location	Status
Arbitrage Opportunity	Medium	DecreasePositionCollateralUtils.sol: 126	Unresolved

Description

For markets where the `pnlToken` can be the same as the `indexToken`, there are cases where the `indexToken` and `pnlToken` are valued differently and users benefit from this difference at the market's expense.

Consider a `StopLossDecrease` order for a long in profit, the `pnlToken` is the same as the `indexToken`.

The user's pnl calculation can value the `indexToken` at the `acceptablePrice`, say $5495. However the resulting `values.positionPnlUsd` is converted to the `pnlToken` at the `secondaryPrice`, say $5490.

This yields a delta of tokens that was not originally factored into the PnL for the market, so the market experiences slightly more loss than expected and the user gains slightly more than expected.

Recommendation

For cases where the `pnlToken` is the same as the `indexToken`, consider using the `executionPrice` to denominate the `values.pnlAmountForPool`.

DPU-2 | Drain Keeper's Gas Through Liquidations

Category	Severity	Location	Status
Gas Attack	Medium	DecreasePositionUtils.sol: 162, 218	Unresolved

Description PoC

A trader is allowed to decrease their position such that the collateral is below the minimum collateral because `shouldValidateMinCollateralUsd` is `false`. However, `shouldValidateMinCollateralUsd` is set to `true` for liquidation orders.

Therefore, a trader's decrease order can go through and their position can be liquidated right after by a liquidation keeper. An attacker may leverage this to drain the keeper of its gas by creating trivial positions and decreasing them to invalidate the minimum collateral so that they are subsequently liquidated.

Recommendation

Always validate the minimum collateral amount when decreasing or increasing a position.

PPU-3 | No Lower Bound On Virtual Inventory Price Impact

Category	Severity	Location	Status
Logical Error	Medium	PositionPricingUtils.sol: 217	Unresolved

Description

In the `getPriceImpactUsd` function the `priceImpactUsdForVirtualInventory` is asserted to be `<=` the `thresholdPriceImpactUsd`, otherwise the normal `priceImpactUsd` is used.

This however allows the `priceImpactUsdForVirtualInventory` to negatively impact users without bound. This way malicious actors in other markets are able to grief users using the same `priceImpactUsdForVirtualInventory`.

Recommendation

Modify the `priceImpactUsdForVirtualInventory > thresholdPriceImpactUsd` comparison to `priceImpactUsdForVirtualInventory < thresholdPriceImpactUsd`.

PPU-4 | borrowingFeeAmountForFeeReceiver Double Counted

Category	Severity	Location	Status
Double Counting	Medium	PositionPricingUtils.sol: 396	Unresolved

Description

The `fees.totalNetCostAmount` includes both the `fees.feeReceiverAmount` and the `fees.borrowingFeeAmount`. The `borrowingFeeAmount` is comprised of both the borrowing fees for the pool and for the `feeReceiver`.

The `fees.feeReceiverAmount` also includes the borrowing fees for the `feeReceiver`, therefore the `borrowingFeeAmountForFeeReceiver` amount is accounted for twice in the `fees.totalNetCostAmount`.

Recommendation

Only account for the `borrowingFeeAmountForFeeReceiver` once in the `fees.totalNetCostAmount`.

MKTU-8 | Precision Loss For Funding Fees

Category	Severity	Location	Status
Precision Loss	Medium	MarketUtils.sol: 891	Unresolved

Description

When computing the `cache.fundingUsd`, a USD amount with 30 decimals of precision is divided by 1e30: `cache.sizeOfLargerSide / Precision.FLOAT_PRECISION`.

This results in precision loss on the order of magnitude of tens of cents for the distribution of funding fees.

Recommendation

Consider if this magnitude of precision loss is acceptable and adjust the calculation if it isn't.

SPRU-1 | Double Counting Swap Imbalance

Category	Severity	Location	Status
Double Counting	Medium	SwapPricingUtils.sol: 109	Unresolved

Description

When calculating the price impact USD value for a swap, the `thresholdPriceImpactUsd` is based on the `params.usdDeltaForTokenA.abs()` and `params.usdDeltaForTokenB.abs()`. However these values will always have the same magnitude during a swap, therefore the user's swap USD value will be double counted.

If the configured `thresholdImpactFactorForVirtualInventory` is intended to be 70% of the user's swap USD value, it will instead account for 140% of the user's swap USD value.

Recommendation

Only base the `thresholdPriceImpactUsd` on a single token side of `usdDelta` for swaps.

DPU-3 | Position Unexpectedly Closed

Category	Severity	Location	Status
Unexpected Behavior	Medium	DecreasePositionUtils.sol: 143	Unresolved

Description

In the case where a user's collateral is deemed to be insufficient after removing the `initialCollateralDeltaAmount`, the `initialCollateralDeltaAmount` is set to 0.

However if the `estimatedRemainingCollateralUsd`, which was based upon the `initialCollateralDeltaAmount` being removed from the position's collateral, is smaller than the `MIN_COLLATERAL_USD` the position will still be closed.

This is unexpected behavior as the `initialCollateralDeltaAmount` has been set to 0 so the position's collateral will no longer be less than the `MIN_COLLATERAL_USD`.

Recommendation

Do not close the user's position in the case where the `initialCollateralDeltaAmount` is set to 0.

GLOBAL-7 | Funding Fees Accumulate In Disabled Markets

Category	Severity	Location	Status
Unexpected Behavior	Medium	Global	Unresolved

Description

If increase and decrease position functionalities are disabled, funding fees will continue to accumulate as time goes by. Traders will be charged unexpected fees when trading resumes.

Recommendation

Consider pausing funding fee accumulation when trading is disabled.

ADLU-2 | Latest ADL Block Updated Without ADL State Change

Category	Severity	Location	Status
Logical Error	Medium	AdlUtils.sol: 114	Unresolved

Description

The latest ADL block is updated each time `updateAdlState` is called even if the ADL state was not enabled or changed. This may lead to a keeper continuously updating ADL block and preventing much needed ADL decrease orders from going through.

Recommendation

Consider using `if (shouldEnableAdl) setLatestAdlBlock(dataStore, market, isLong, Chain.currentBlockNumber())`.

Otherwise, if current functionality is intended, add more documentation surrounding ADL block updates.

ORDH-4 | Risk-Free Trade With Disabled Feature

Category	Severity	Location	Status
Protocol Manipulation	Medium	OrderHandler.sol: 247	Unresolved

Description

If a limit order is in the `dataStore` and the trading features are disabled, then the possibility of a risk-free trade arises.

Right before a feature is re-enabled, if prices have moved against the trader, the trader may cancel or update their limit order. Otherwise, the order can execute with outdated prices.

Recommendation

Do not revert on the `FeatureUtils.DisabledFeature` error, but rather freeze or cancel these limit orders.

DPCU-5 | Remaining Collateral Adjusted To Revert

Category	Severity	Location	Status
Logical Error	Medium	DecreasePositionCollateralUtils.sol: 236, 239	Unresolved

Description

When `values.remainingCollateralAmount` is less than or equal to `collateralCache.adjustedPriceImpactDiffAmount`, `values.remainingCollateralAmount` is set to 0 and the `collateralCache.adjustedPriceImpactDiffAmount` is placed in the holding area.

The position is stamped with a collateral amount of 0 in `DecreasePositionUtils.sol` on line 195. Once `validatePosition` is entered, `validateNonEmptyPosition` will be called which will revert due to the 0 collateral amount with the `EmptyPosition` error.

Overall, the position's `remainingCollateral` is updated only to subsequently revert when `validatePosition` is called. Furthermore, this may open the market to scenarios where a user can influence when their order is executed and create a risk-free trade as the order stays in the store with the same `updatedAtBlock`.

Recommendation

Do not set the `remainingCollateralAmount` to 0 only to have the order revert later on with an `EmptyPosition` error. Instead consider leaving some `remainingCollateral` or reverting with a separate error so that the order is cancelled or frozen.

WTDU-2 | Swap Required To Specify Output Amount

Category	Severity	Location	Status
Slippage	Medium	WithdrawalUtils.sol: 405	Unresolved

Description

A liquidity provider is unable to utilize the `minOutputAmount` functionality without performing a swap. This may lead LPs to get less output than intended and/or confusion with the existing `minLongTokenAmount` and `minShortTokenAmount` usage in swap.

Recommendation

Consider adding the functionality to specify the minimum output amounts without swapping.

GLOBAL-8 | Lack of Slippage Protection When Swapping

Category	Severity	Location	Status
Slippage	Medium	Global	Unresolved

Description

The following places lack slippage protection and may lead to loss of assets for a user:

1) `DecreasePositionCollateralUtils.sol` Line 392 and Line 434 use 0 as the `minOutputAmount` which may unexpectedly reduce the profit for a trader.
2) `ExecuteDepositUtils.sol` Line 432 which may reduce the amount of market tokens the user obtains although risk may be limited by specifying `minMarketTokens`.

Recommendation

Consider adding the functionality to specify the minimum output amounts and/or further document this behavior.

GSU-3 | No Way For Users To Claim Excess Execution Fee

Category	Severity	Location	Status
Lost Funds	Medium	GasUtils.sol: 88	Unresolved

Description

`handleExcessExecutionFee` does not allow users to claim the excess `executionFee` that they may have sent. Currently the excess is simply sent to a holding address with no accounting of which user is in excess or by how much and there is no way to claim the excess fee.

Recommendation

Either make a way for users to claim these tokens or ensure it is well documented and explicit that these tokens will be lost for the user.

ADLU-3 | Two Separate ADL Factors

Category	Severity	Location	Status
Logical Error	Medium	AdlUtils.sol: 110	Unresolved

Description

`Keys.MAX_PNL_FACTOR` is used when checking if the PnL factor for ADL is exceeded instead of `Keys.MAX_PNL_FACTOR_FOR_ADL` as in `AdlHandler.sol` line 123. This can lead to ADL being enabled and not going through, or ADL being consistently disabled due to misconfiguration between the two factors.

Recommendation

Use the same factor key for the same validation. If the difference is intended for finer and more precise protocol control, document such behavior.

SWPU-2 | Swaps Prevented When They Improve The Pool

Category	Severity	Location	Status
Logical Error	Medium	SwapUtils.sol: 291	Unresolved

Description

When performing a swap, `Keys.MAX_PNL_FACTOR_FOR_WITHDRAWALS` is used to check whether the current `pnlToPoolFactor` exceeds the the maximum PnL factor for withdrawals, which is the strictest (lowest) maximum `pnlToPoolFactor`.

When performing a swap, the `tokenIn` is deposited and `tokenOut` is withdrawn. By treating both the "deposit" and "withdrawal" with the same withdrawal `pnlToPoolFactor` threshold, it can potentially prevent swaps that will improve the current `pnlToPoolFactor` on a particular side.

Recommendation

Consider validating `tokenIn` against the `MAX_PNL_FACTOR_FOR_DEPOSITS` and `tokenOut` against `MAX_PNL_FACTOR_FOR_WITHDRAWALS`.

ORDH-5 | Gas Used Is Overestimated

Category	Severity	Location	Status
Logical Error	Medium	OrderHandler.sol: 178	Unresolved

Description

When passing the `startingGas` to the `this._executeOrder` external call, it is assumed that all of the `startingGas` is available for the execution of the external call. However, due to the 63/64 rule, only 63/64 of the `startingGas` will be available in the subsequent call to `this._executeOrder`.

Therefore when the `executionFee` is paid in the external call to `this._executeOrder`, the gas used will be overestimated, and the user will be errantly charged for a false 1/64 expenditure.

Recommendation

Account for the 63/64 rule when estimating the gas consumption.

GLOBAL-9 | Pool State Leading To Withdrawals Being Bricked

Category	Severity	Location	Status
Trapped Funds	Medium	Global	Unresolved

Description PoC

With a range between the `MAX_PNL_FACTOR_FOR_WITHDRAWALS` and the `MAX_PNL_FACTOR_FOR_ADL`, if the profit in the pool exceeds the `pnlToPoolFactor` for withdrawals but is not high enough to trigger ADL, withdrawals for LPs will be bricked until users deposit more tokens into the pool and a trader decides to close their profitable position.

Recommendation

Ensure that this risk is well communicated. Additionally consider setting the `MAX_PNL_FACTOR_FOR_WITHDRAWALS` close to the `MAX_PNL_FACTOR_FOR_ADL` to limit this scenario.

EDPU-3 | Market Token Price Below Allowed Amount

Category	Severity	Location	Status
Logical Error	Medium	ExecuteDepositUtils.sol: 122	Unresolved

Description PoC

Deposits are restricted if the `MAX_PNL_FACTOR_FOR_DEPOSITS` is exceeded since the market token price is below the “allowed” amount. However, market token price can continue to decrease after ADL, and deposits can once again resume.

Recommendation

Do not count on this minimum bound for market token price and be sure to document that the price can keep dropping below the asserted “allowed” amount.

CLC-1 | boundedSub Can Underflow

Category	Severity	Location	Status
Logical Error	Medium	Calc.sol: 116	Unresolved

Description

The `boundedSub` function does not correctly prevent underflow.

For example, `boundedSub(type(int256).min, 1)` causes an underflow and reverts because the condition check `if (a < 0 && b <= type(int256).min - a)` is incorrect.

This poses inherent risk for the future use of `boundedSub` in this codebase and others that may adopt it.

Recommendation

Replace with `if (a < 0 && b >= a - type(int256).min)` or `if (a < 0 && -b <= type(int256).min - a)`.

GLOBAL-10 | Double Fee May Make A Position Liquidatable

Category	Severity	Location	Status
Double Counting	Medium	Global	Unresolved

Description PoC

When increasing or decreasing a position, fees are calculated based on the size of the order and then taken out of the collateral.

However, when validating the position with `validatePosition` and checking `isPositionLiquidatable`, fees are calculated and applied a second time. This further reduces how much collateral the position has during this validation. This can prevent increasing or decreasing a position as the `isPositionLiquidatable` check would fail unexpectedly.

Recommendation

Check if the position is liquidatable prior to fees being paid and taken out of the collateral.

PPU-5 | Borrowing Fees Maximized Twice

Category	Severity	Location	Status
Logical Error	Low	PositionPricingUtils.sol: 351	Unresolved

Description

When updating the `cumulativeBorrowingFactor`, the fees are maximized in the protocol's favor as can be seen in `getBorrowingFactorPerSecond`. The `poolUsd` uses the `min` price and the `reservedUsd` uses the `max` price.

However, in `getPositionFees`, the amount of collateral tokens is calculated from the borrowing fees using the `min` price. Then, that amount will be multiplied by the `max` price of the collateral token. Therefore the resulting borrowing fee is maximized twice. This means the borrowing fees will be larger than expected, and the position more likely to be liquidated.

An explicative example:

- A collateral token has a min-max range of $1-$2.
- The `borrowingFee` is $100 which is already maximized.
- The resulting `fees.borrowingFeeAmount` is $100 / $1 = 100 collateral tokens.
- The `fees.totalNetCostUsd` calculation multiplies 100 * $2 = $200 in borrowing fees.

Therefore, what was originally a $100 fee turns into a $200 fee.

Recommendation

Use the direct value returned from the `getBorrowingFactorPerSecond` function for the `fees.totalNetCostUsd`.

POSU-2 | Unexpected Closing Of Positions

Category	Severity	Location	Status
Logical Error	Low	PositionUtils.sol: 401	Unresolved

Description

The `minCollateralFactor` is derived from the current open interest such that the remaining collateral of a position is positively correlated with open interest.

E.g. as there is more open interest in a market, more remaining collateral is required to be considered sufficient. This prevents smaller positions from being opened in markets with high open interest.

Additionally, users who originally opened positions in markets before the open interest had grown may find themselves unable to decrease their position collateral without closing their entire position.

Recommendation

Reconsider if these externalities are desired and if so make sure they are well documented.

POSU-3 | Swapped Parameters

Category	Severity	Location	Status
Typo	Low	PositionUtils.sol: 459, 460	Unresolved

Description

`params.position.borrowingFactor()` and `params.position.sizeInUsd()` are swapped as `MarketUtils.updateTotalBorrowing` takes the `prevPositionSizeInUsd` followed by the `prevPositionBorrowingFactor`. The end result is the same because multiplication is commutative, but poses a risk for future changes.

Recommendation

Flip the parameters so they are correctly ordered.

WTDU-3 | Overflow Risk

Category	Severity	Location	Status
Overflow	● Low	WithdrawalUtils.sol: 456-457	Unresolved

Description

When making a withdrawal, the `_getOutputAmounts` function multiplies two float precision USD amounts. This multiplication can result in overflow when both USD amounts are on the order of hundreds of millions. E.g. $600,000,000 * $200,000,000 will overflow and revert.

Recommendation

Be aware of this overflow risk, and consider altering the arithmetic if values of this size are expected.

DPU-4 | Affiliate Rewards Upon Liquidation

Category	Severity	Location	Status
Incentives	Low	DecreasePositionUtils.sol: 261	Unresolved

Description

Affiliates still receive their `fees.referral.affiliateRewardAmount` upon liquidation because the `handleReferral` function is always called in the `decreasePosition` function.

Recommendation

Consider whether this is desired behavior. If not, do not call the `handleReferral` in the case of liquidations.

DPU-5 | Worst Case Estimation Unused

Category	Severity	Location	Status
Validation	Low	DecreasePositionUtils.sol: 103	Unresolved

Description

`cache.prices.indexTokenPrice.midPrice()` is used to estimate the position's PnL for `cache.estimatedPositionPnlUsd`.

However, it may make more sense to use `prices.indexTokenPrice.pickPriceForPnl` as in `isPositionLiquidatable` to get the worst-case scenario price for estimation purposes.

Recommendation

Consider switching `cache.prices.indexTokenPrice.midPrice()` to `prices.indexTokenPrice.pickPriceForPnl(isLong, false)`.

POSU-4 | Funding Fees Sent To Receiver

Category	Severity	Location	Status
Unexpected Behavior	Low	PositionUtils.sol: 477, 488	Unresolved

Description

In the `incrementClaimableFundingAmount` function, the claimable funding fees are incremented for the receiver rather than the position owner. This could be unexpected and may lead to loss of funds in the event that the receiver is a contract.

Recommendation

Consider if the `position.account()` should receive the claimable funding fees rather than the `params.order.receiver()`.

PPU-6 | Sandwich Attack For Price Impact

Category	Severity	Location	Status
Protocol Manipulation	● Low	PositionPricingUtils: 195	Unresolved

Description

A trader may perform a weak version of a sandwich attack on large `LimitIncrease` orders to benefit from the `priceImpact` in the following way:

- A malicious trader observes that the `triggerPrice` is approaching for a `LimitIncrease` order.
- The malicious trader creates a `MarketIncrease` order that will balance the open interest in the pool to receive positive price impact.
- The `LimitIncrease` order is executed and gets negatively price impacted because it unbalances the pool.
- The malicious trader creates a `MarketDecrease` order that will rebalance the pool and receive positive impact.

Recommendation

There are levers in place to limit the scope of these sandwich attacks such as the two-step execution system and the `acceptablePrice`, but nonetheless the risk of such an attack should be well documented.

IOU-1 | Limit Increase With Swap Path May Be Griefed

Category	Severity	Location	Status
Protocol Manipulation	Low	IncreaseOrderUtils.sol: 26	Unresolved

Description PoC

Malicious traders can stop `LimitIncrease` orders from getting filled if the order has a `swapPath`.

A malicious trader can observe that the `triggerPrice` is approaching for the `LimitIncrease` and then create their own `MarketSwap` that removes the necessary liquidity to execute the `swapPath` of the `LimitIncrease` order.

Recommendation

Be aware and document that user's `LimitIncrease` orders can be fail due to their `swapPath`.

SWPU-3 | Event Getting Incorrect Value

Category	Severity	Location	Status
Logical Error	Low	SwapUtils.sol: 303	Unresolved

Description

In the case where the swap is negatively impacted, the `cache.amountIn` includes the `negativeImpactAmount`. This misrepresents the `amountInAfterFees` in the `emitSwapInfo` function.

Recommendation

Do not include negative impact in the `amountInAfterFees` that is provided to the `emitSwapInfo` function.

DPCU-6 | Lost Funding Fees

Category	Severity	Location	Status
Lost Funds	● Low	DecreasePositionCollateralUtils.sol: 350	Unresolved

Description

In the event that a user's position is liquidated with negative collateral, an empty `PositionFees` object is returned from the `getLiquidationValues` function. This means that the `claimableLongTokenAmount` and `claimableShortTokenAmount` are reset to 0.

In this case any funding fees that the trader had accumulated are lost.

Recommendation

Consider if this is the expected behavior. If not, maintain the existing `claimableLongTokenAmount` and `claimableShortTokenAmount` in the new `PositionFees` object returned from `getLiquidationValues`.

TIME-1 | Bespoke Key Used

Category	Severity	Location	Status
Logical Error	● Low	Timelock.sol: 172, 209	Unresolved

Description

The action key `signalSetPriceFeed` key should be `setPriceFeed` to match up with the other key patterns.

Additionally, when performing the `_validateAndClearAction`, the label when validating & clearing should be `setPriceFeedAfterSignal` on line 209 as well.

Recommendation

Update the keys as recommended.

CLC-2 | RoundUpDivision Rounds Down Instead Of Up

Category	Severity	Location	Status
Documentation	Low	Calc.sol: 36	Unresolved

Description

The `roundUpDivision` function rounds down instead of up when `a` is negative.

Recommendation

Add documentation that this function rounds up purely the magnitude of the integer `a`.

ORDH-6 | Limit Order Cancellation Logic

Category	Severity	Location	Status
Protocol Manipulation	Low	OrderHandler.sol	Unresolved

Description

When markets are disabled, limit orders are not cancelled but frozen. This opens the opportunity for the frozen order keeper to possibly execute orders with outdated prices when a market is re-enabled. The use of outdated prices could lead to risk-free trade opportunities.

Recommendation

Consider adding cancellation logic for limit orders that revert due to disabled markets.

OCLU-1 | Misnamed Variable

Category	Severity	Location	Status
Readability	Low	OracleUtils.sol: 225	Unresolved

Description

The `blockNumber` variable represents a timestamp value rather than a block number value.

Recommendation

Rename variable to something more fitting.

GLOBAL-11 | Additional Feature Controls

Category	Severity	Location	Status
Controls	Low	Global	Unresolved

Description

It may be prudent to add features that can be disabled for things like the `DecreasePositionSwapTypes`.

Recommendation

Consider adding additional features that can be disabled.

SWPU-4 | Missing Check For tokenIn

Category	Severity	Location	Status
Validation	Low	SwapUtils.sol	Unresolved

Description

It is possible to execute a swap with 0 `tokenIn` and a non-zero `swapPath` which is just a waste of resources.

Recommendation

Consider adding `tokenIn > 0` as a validation.

OCL-1 | Unsorted Max Oracle Block Numbers

Category	Severity	Location	Status
Validation	● Low	Oracle.sol	Unresolved

Description

It is possible for the max oracle block numbers to be unsorted unlike the min oracle block numbers. This is because the only requirement for the max block numbers is that they are at least as large as the min oracle block numbers.

For example, the keeper may pass:

min block #'s: [5,5]
max block #'s [6, 5]

Recommendation

Validate that the max oracle block numbers are ascending.

PREC-1 | SafeCast Revert

Category	Severity	Location	Status
Documentation	Low	Precision.sol: 110	Unresolved

Description

Calculating the `result` will be safe as long as `value` is `<= 10^47`. However, the `result` may not fit in a `int256` leading to a revert `SafeCast: value doesn't fit in an int256`

For example, the inputs 57923633301321315440238040719798086540604777067 and 1 yield a `SafeCast` revert.

Recommendation

Document such behavior.

BOU-1 | setExactOrderPrice Using Older Price

Category	Severity	Location	Status
Logical Error	Low	BaseOrderUtils.sol: 227	Unresolved

Description

In the event that the oracle provides multiple prices for a single token during the execution of a market order or a liquidation, the `customPrice` assigned in `setExactOrderPrice` (`primaryPrice`) would differ from the price retrieved and used from `getMarketPricesForPosition` (`secondaryPrice`).

The price used to validate liquidation (`secondaryPrice`) and the price used to execute the liquidation (`primaryPrice`) would be different.

Recommendation

Consider using the `secondaryPrice` for market orders in the event that a range is errantly provided.

BNK-1 | Future Proof Receive Function

Category	Severity	Location	Status
Future Proofing	● Low	Bank.sol	Unresolved

Description

Popular wrapped network tokens like WAVAX and WFTM which the synthetics exchange will likely interact with use `.transfer` to withdraw native tokens to the caller.

The Bank contract's receive function does not currently consume more than 2300 gas, but gas consumption for opcodes like SLOAD which are being used in the receive function are subject to change over time which may cause the receive function to revert when called with a `.transfer`.

(See Berlin Hardfork)

In such a scenario the exchange would be unable to withdraw it's WNT balance into NT and the receive failure could open up opportunities for risk free trading exploits.

Recommendation

Be wary of this possibility and have a plan in the event that gas costs are updated.

DPU-6 | Inaccurate Fees May Be Emitted

Category	Severity	Location	Status
Events	Low	DecreasePositionUtils: 263	Unresolved

Description

The `fees.totalNetCostAmount` is misrepresented in `emitPositionFeesCollected` since the fee is reduced by the profit amount in the `processCollateral` function.

Recommendation

Account for the fees being reduced by profits in the event.

OCL-2 | Missing Check

Category	Severity	Location	Status
Validation	Low	Oracle.sol: 454-456	Unresolved

Description

It is checked that the `oracleTimestamp` was set not too far in the past but there is no check to ensure that the `oracleTimestamp` is not set in the future.

Recommendation

Add a check that validates that the `oracleTimestamp` is not in the future.

DPU-7 | Collateral May Not Be Sufficient

Category	Severity	Location	Status
Logical Error	Low	DecreasePositionUtils.sol: 139	Unresolved

Description

The position's collateral still may not be considered sufficient even after the `initialCollateralDeltaAmount` is set to 0.

Recommendation

Consider whether the order should still be allowed to execute in the case where the position's collateral is still not sufficient even when the `initialCollateralDeltaAmount` is 0. If not, revert with an error.

MKTU-9 | Duplicated Code

Category	Severity	Location	Status
Optimization	Low	MarketUtils.sol: 1802, 1817	Unresolved

Description

The `validateEnabledMarket` function can be reused once the market is obtained from the address instead of duplicating code.

Recommendation

Consider implementing the above suggestion.

PPU-7 | Variable Reuse

Category	Severity	Location	Status
Optimization	Low	PositionPricingUtils.sol: 396	Unresolved

Description

The `fees.feeAmountForPool` can be reused in the `fees.totalNetCostAmount` calculation.

Recommendation

Consider implementing the above suggestion.

POSU-5 | Use Cheaper Branch

Category	Severity	Location	Status
Gas Optimization	Low	PositionUtils.sol: 336	Unresolved

Description

The `priceImpactUsd > 0` check can include 0 to save gas from the `else` case.

Recommendation

Update the conditional to `priceImpactUsd >= 0`.

GLOBAL-12 | Internal Library Functions

Category	Severity	Location	Status
Visibility Modifiers	Low	Global	Unresolved

Description

Library functions throughout could be made `internal` to save gas.

Recommendation

Make as many library functions `internal` as possible while staying within contract bytecode deployment limits.

DPCU-7 | Use Cached Variable

Category	Severity	Location	Status
Optimization	Low	DecreasePositionCollateralUtils.sol: 95	Unresolved

Description

The `initialCollateralDeltaAmount` variable is left unused and the attribute is instead repeatedly accessed directly from the order.

Recommendation

Use the cached `initialCollateralDeltaAmount` or remove it.

POSU-6 | Function Reuse

Category	Severity	Location	Status
Optimization	Low	PositionUtils.sol: 220	Unresolved

Description

The `sizeDeltaUsd` calculation for shorts can be replaced with the `getSizeDeltaInTokens` function.

Recommendation

Consider implementing the above suggestion.

GLOBAL-13 | Initialization of Default Values

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

Throughout the codebase the index for many for-loops is initialized to 0 which is the default `uint` value. Avoid the unnecessary initialization and allow the default values to be implicitly assigned to these `uint` variables:

- FeeHandler.sol::37
- MarketUtils.sol::1859
- Oracle.sol::235
- Oracle.sol::287
- Oracle.sol::440
- Oracle.sol::476
- Oracle.sol::494
- Oracle.sol::562
- OracleModule.sol::61
- OracleModule.sol::67
- OracleUtils.sol::195
- ExchangeRouter.sol::283
- ExchangeRouter.sol::309

Recommendation

Do not assign these default values.

GLOBAL-14 | Initialization of Default Values

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

Throughout the codebase the index for many for-loops is initialized to 0 which is the default `uint` value. Avoid the unnecessary initialization and allow the default values to be implicitly assigned to these `uint` variables:

- ExchangeRouter.sol::343
- SwapUtils.sol::123
- Array.sol::56
- Array.sol::73
- Array.sol::90
- Array.sol::107
- Array.sol::124
- BasicMulticall.sol::17
- PayableMulticall.sol::21

Recommendation

Do not assign these default values.

GLOBAL-15 | Cached Array Length

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the codebase there are many for-loops that compute the length of an array upon each iteration. Caching the length of these arrays will decrease gas costs throughout the application:

- FeeHandler.sol::37
- MarketUtils.sol::1859
- Oracle.sol::235
- Oracle.sol::287
- Oracle.sol::440
- Oracle.sol::476
- Oracle.sol::494
- Oracle.sol::562
- OracleModule.sol::61
- OracleModule.sol::67
- OracleUtils.sol::195

Recommendation

Cache the length of arrays before iterating through them.

GLOBAL-16 | Cached Array Length

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the codebase there are many for-loops that compute the length of an array upon each iteration. Caching the length of these arrays will decrease gas costs throughout the application:

- ExchangeRouter.sol::283
- ExchangeRouter.sol::309
- ExchangeRouter.sol::343
- SwapUtils.sol::123
- Array.sol::56
- Array.sol::73
- Array.sol::90
- Array.sol::107
- Array.sol::124
- BasicMulticall.sol::17
- PayableMulticall.sol::21

Recommendation

Cache the length of arrays before iterating through them.

GLOBAL-17 | Greater Than Zero Check

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the codebase `uint` variables are checked to be `> 0` however `!= 0` is a more efficient comparison to check if `uint` values are positive. Switching to `!= 0` will decrease gas costs throughout the application:

- ExecuteDepositUtils.sol::163
- ExecuteDepositUtils.sol::192
- ExecuteDepositUtils.sol::208
- GasUtils.sol::95
- Oracle.sol::592
- BaseOrderUtils.sol::352
- BaseOrderUtils.sol::369
- DecreaseOrderUtils.sol::57
- DecreasePositionCollateralUtils.sol::174
- DecreasePositionUtils.sol::96
- DecreasePositionUtils.sol::150
- IncreasePositionUtils.sol::151
- IncreasePositionUtils.sol::177
- PositionUtils.sol::471
- PositionUtils.sol::482
- PositionUtils.sol::535
- Precision.sol::61

Recommendation

Use `!= 0` to check if `uint` values are positive.

SWPU-5 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	SwapUtils.sol: 41	Unresolved

Description

The NatSpec documentation for the `SwapParams` is outdated.

Recommendation

Update the NatSpec to reflect the current contents of `SwapParams`.

BOU-2 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	BaseOrderUtils.sol: 34	Unresolved

Description

The `decreasePositionSwapType` is missing from the NatSpec documentation for the `CreateOrderParams` struct.

Recommendation

Update the NatSpec documentation for the `CreateOrderParams` struct.

MKTU-10 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	MarketUtils.sol: 561	Unresolved

Description

The `timeKey` parameter is missing from the NatSpec documentation for the `claimCollateral` function.

Recommendation

Update the NatSpec documentation for the `claimCollateral` function.

POSU-7 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	PositionUtils.sol: 269	Unresolved

Description

The `validatePosition` function does not include `shouldValidateMinCollateralUsd` as an @param in it's NatSpec documentation.

Recommendation

Update the NatSpec for `validatePosition`.

BOU-3 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	BaseOrderUtils.sol: 83	Unresolved

Description

The NatSpec documentation for the `ExecuteOrderParams` is outdated.

Recommendation

Update the NatSpec to reflect the current contents of `ExecuteOrderParams`.

POSU-8 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	PositionUtils.sol: 36	Unresolved

Description

The NatSpec documentation for the `UpdatePositionParams` is outdated.

Recommendation

Update the NatSpec to reflect the current contents of `UpdatePositionParams`.

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