GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

GMX

Summary

Audit Firm Guardian

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

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 18th of April to the 15th of May, 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

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics/tree/v1.3
Commit(s)	f7c02f99c06c3eece92e06bda160e108ab266023

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	3	3	0	0	0	0
High	9	9	0	0	0	0
Medium	14	14	0	0	0	0
Low	17	17	0	0	0	0

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
ADLU	AdlUtils.sol	ee802285ce57808a2bc13c149d39e11dd927018b
BNK	Bank.sol	661c2f7e4227e315febf5800510a25a77f16bb16
SBNK	StrictBank.sol	13276745295cbc093207e92bcb096c9a01e79c99
CBKU	CallbackUtils.sol	e84cdf0526027c47fcb7885ad589efbfb36ca2b3
DCBK	IDepositCallbackReceiver.sol	aed58b8d02e950c17f1e375d1ff4537e20d4d460
OCBK	IOrderCallbackReceiver.sol	b986dcf7d9deb75f6cbb6e630a3f7d2a27f75374
WCBK	IWithdrawalCallbackReceiver.sol	d85b4c126911ed219a4bb13349a35887e9b6db84
ARBS	ArbSys.sol	0d9703a3477e40ccce9b0b526c0c9f4310034496
CHAIN	Chain.sol	020d318af7d3d4ecba2cdf36669c582923611ae9
DATA	DataStore.sol	d0ef37ad3eab38d61c7f1bfa09ad33f0ece6c052
KEY	Keys.sol	c7675da3e1dec9d011800ed49517a892b7542692
DEP	Deposit.sol	9580b364ab6c14e76db4a0058a785cf10264c241
DEPS	DepositStoreUtils.sol	92e262e33d6d1b5c7996d57b7ffcdd5fd25a3b99
EMIT	EventEmitter.sol	3fce680d9fd7432923b859ab7f9fa15e8a96ee14
DEPH	DepositHandler.sol	b6e687590b286b8b1203bcdf85ec3bd5e377659c
ORDH	OrderHandler.sol	9dd2f5bf3b1193ffbfc75158b934c02aca5ec24f
WTDH	WithdrawalHandler.sol	971d901ecdbbbcea00403a024f4bac2bb1894ea3
FTU	FeatureUtils.sol	3ef861b0af29a793b53e10e1cf3ffb39455c81d7
ODV	OrderVault.sol	74f991769825ba9fc8b98f3be3a5fefc32be7539
DPV	DepositVault.sol	1d19ad5afc0baec27a608a2f53cbb5b6f48f8f26

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
WDV	WithdrawalVault.sol	5cc2b331b13f735dfebc983b9aec705692e0d2a2
GSU	GasUtils.sol	ca727248265157cfb3ae892f7c1137dc3f6c984b
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

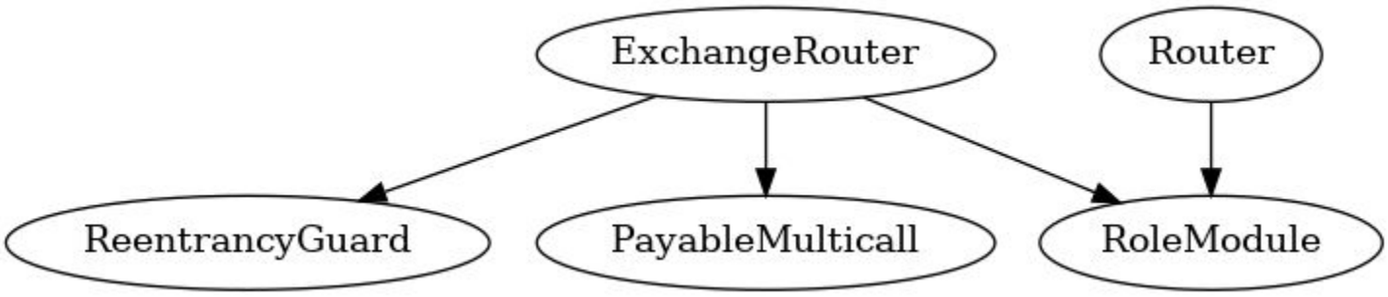
Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract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

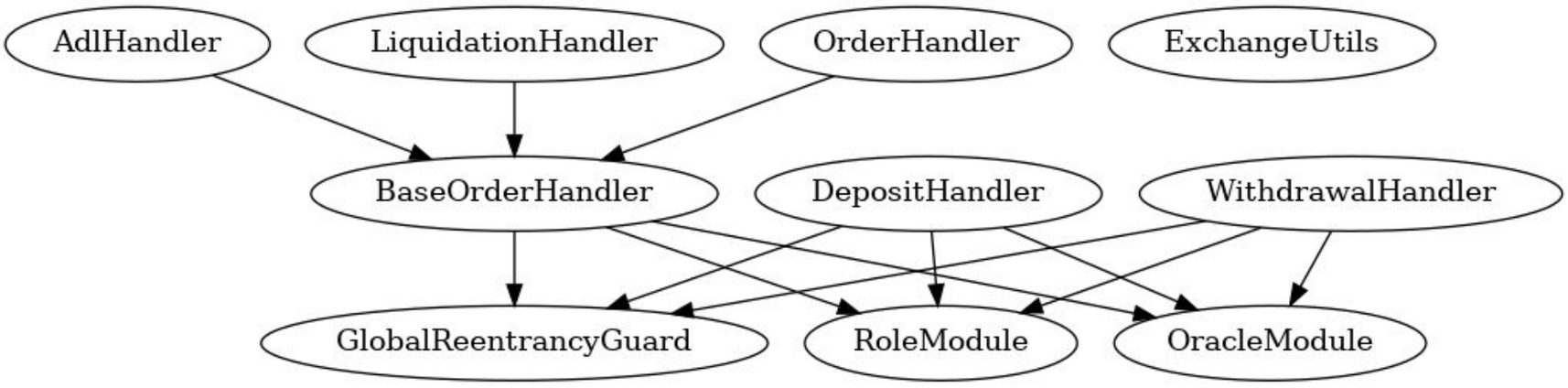
Methodology

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
CON-1	Missing Keys In Config	Configuration	Critical	Unresolved
DPCU-1	Unliquidatable Position Due To getLiquidationValues	Logical Error	Critical	Unresolved
DPCU-2	Mis-accounting When Swap Fails	Logical Error	Critical	Unresolved
MKTU-1	Malicious Actor Can Break Markets	Underflow	High	Unresolved
MKTU-2	Unclaimable Funding Fees	Logical Error	High	Unresolved
TIME-1	Wrong Key For Signer Removal	Logical Error	High	Unresolved
BOU-1	Tight Stop Loss Abuse	Logical Error	High	Unresolved
ERTR-1	UI Fee Manipulation	Protocol Manipulation	High	Unresolved
ORDU-1	Referral Code Manipulation	Protocol Manipulation	High	Unresolved
BOU-2	Stop-loss Won't Execute On Price Gap	Logical Error	High	Unresolved
DPCU-3	Unliquidatable Position Due to PriceImpactDiff	Logical Error	High	Unresolved
BOU-3	Position Impact Pool Manipulation	Protocol Manipulation	High	Unresolved
MKTU-3	Pending Borrowing Fees Brick Withdrawals	Underflow	Medium	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
SWPU-1	Max Price Used For Swap Pricing	Logical Error	Medium	Unresolved
SWOU-1	LimitSwaps Unnecessarily Delayed	Logical Error	Medium	Unresolved
DOU-1	Minimum Output Amount Grieifing	Griefing	Medium	Unresolved
KEY-1	Wrong Key For Pool Adjustment	Typo	Medium	Unresolved
IPU-1	Price Impact Double Counted	Double Counting	Medium	Unresolved
DPU-1	Token Amount Added To USD Value	Logical Error	Medium	Unresolved
EDPU-1	Users Are Negatively Affected By The Price Spread	Logical Error	Medium	Unresolved
GLOBAL-1	Liquidations When Features Disabled	Logical Error	Medium	Unresolved
OCL-1	Lack Of Sequencer Uptime Check	Validation	Medium	Unresolved
CHAIN-1	Hardcoded Chain ID	Configuration	Medium	Unresolved
POSU-1	Negative PnL Ignored In Sufficient Collateral Check	Validation	Medium	Unresolved
GLOBAL-2	Block Re-org Attack	Block Re-org	Medium	Unresolved
GLOBAL-3	Multiple Read-only Reentrancies	Reentrancy	Medium	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-4	Minimum Order Size	Validation	Low	Unresolved
BOU-4	Superfluous positionKey Variable	Superfluous Code	Low	Unresolved
GLOBAL-5	Revert Reason Unnecessarily Parsed	Optimization	Low	Unresolved
GLOBAL-6	validateMarketTokenBalance After External Call	Validation	Low	Unresolved
ERR-1	Superfluous Error	Superfluous Code	Low	Unresolved
EDPU-2	Recomputed Value	Optimization	Low	Unresolved
PPU-1	Outdated NatSpec	Documentation	Low	Unresolved
GLOBAL-7	ERC-777 Tokens	Warning	Low	Unresolved
PPU-2	Misleading Comment	Documentation	Low	Unresolved
GLOBAL-8	Invalid Market Risk	Warning	Low	Unresolved
OCL-2	Misleading Comment	Documentation	Low	Unresolved
MKTU-4	Prefer applyFactor	Precision	Low	Unresolved
GLOBAL-9	Superfluous Code	Superfluous Code	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
ORDU-2	Duplicate Validation	Superfluous Code	Low	Unresolved
KEY-2	Outdated NatSpec	Documentation	Low	Unresolved
DS-1	Errant Import	Superfluous Code	Low	Unresolved
OCL-3	Direct Use Of block.timestamp	Consistency	Low	Unresolved

CON-1 | Missing Keys In Config

Category	Severity	Location	Status
Configuration	● Critical	Config.sol	Unresolved

Description

Several critical keys are missing from the `initAllowedBaseKeys` function:

- `MIN_POSITION_SIZE_USD`
- `MAX_PNL_FACTOR_FOR_DEPOSITS`
- `MAX_PNL_FACTOR_FOR_ADL`

Recommendation

Add the missing keys to `initAllowedBaseKeys`.

DPCU-1 | Unliquidatable Position Due To getLiquidationValues

Category	Severity	Location	Status
Logical Error	● Critical	DecreasePositionCollateralUtils.sol: 367	Unresolved

Description PoC

In the `getLiquidationValues` function the `values.pnlAmountForPool` is reset to a new value, although the previous value may have been used in a swap from the `pnlToken` to the `collateralToken`.

This causes mis-accounting in the market and causes a revert with the market token balance check upon liquidation. Therefore positions can be unliquidatable, yielding a potentially catastrophic amount of bad-debt for the market.

Recommendation

The solution is to account for the previous `values.pnlAmountForPool` in the event that this swap was made. e.g. add:

```
if (wasSwapped) {
   MarketUtils.applyDeltaToPoolAmount(
       params.contracts.dataStore,
       params.contracts.eventEmitter,
       params.market.marketToken,
       values.pnlTokenForPool,
       values.pnlAmountForPool
   );
}
```

to the `else` branch in `getLiquidationValues`.

DPCU-2 | Mis-Accounting When Swap Fails

Category	Severity	Location	Status
Logical Error	● Critical	DecreasePositionCollateralUtils.sol: 141, 188, 221, 224	Unresolved

Description

Positions in profit with unpaid borrowing/funding fees that are greater than the position's collateral open the exchange up to several high-impact issues when the `swapProfitToCollateralToken` swap fails. These positions are able to exist since the `isPositionLiquidatable` check factors positive PnL as collateral that would purportedly always be able to cover these fees.

However `swapProfitToCollateralToken` will commonly fail whenever the `validatePoolAmount`, `validateReserve`, or `validateMaxPnl` checks fail as a result of the swap, causing the following issues:

- Positions in large profit would be un-ADL-able since the ADL order would revert on line 224 as the collateral is not sufficient to cover the fees alone.
- Liquidations for these positions result in the user losing all of their profit since the execution enters `getLiquidationValues`.
- Liquidations for these positions result in the protocol having to cover a potentially large deficit between the position's collateral and the unpaid funding fees.
- The pool value for market depositors sees a stepwise jump down from the potentially large unpaid borrowing fees.
- Decrease orders expecting to be able to use their profit to pay fees will be cancelled/frozen

Recommendation

Do not allow positions to exist when their fees are greater than the actual collateral backing the position. Allow positions to be liquidated when their fees negate the collateral backing amount.

MKTU-1 | Malicious Actor Can Break Markets

Category	Severity	Location	Status
Underflow	High	MarketUtils.sol: 332	Unresolved

Description PoC

The `uint poolValue` is decreased by the impact pool value before the PnL is added to the `poolValue`. Additionally, the impact pool value is not capped to avoid underflow.

Because of this, there are some cases where the market can be entirely bricked when the value of the impact pool surpasses the value of the backing tokens — even if it was meant to offset a positive pool PnL.

A malicious actor can engineer this outcome in certain scenarios, especially when the pool is initially deployed.

Recommendation

Consider moving the `poolValue -= result.impactPoolAmount * indexTokenPrice.pickPrice(maximize);` line to after the PnL is added to the `poolValue`, as the `impactPoolAmount` is meant to offset initial positive pnl. Otherwise, consider making the `poolValue` an `int` within the `getPoolValueInfo` function.

Additionally, consider capping the value of the impact pool (similarly to the capping of PnL) that is subtracted from the `poolValue` to avoid any cases where the market is bricked.

MKTU-2 | Unclaimable Funding Fees

Category	Severity	Location	Status
Logical Error	High	MarketUtils.sol: 2368, 2369	Unresolved

Description

In the `getExpectedMinTokenBalance` function, the `collateralForLongs` and `collateralForShorts` is included in the resulting `expectedMinBalance`.

Therefore users who are paid out funding fees will be unable to claim them until the users who are paying the funding fees update their position. There is no requirement for users to frequently update their position and therefore funding fees can go a long time without being claimable for users.

Recommendation

Adjust `getExpectedMinTokenBalance` such that funding fees that will be paid from user's collateral can be immediately claimed without affecting the validation.

TIME-1 | Wrong Key For Signer Removal

Category	Severity	Location	Status
Logical Error	High	Timelock.sol: 118	Unresolved

Description

When a remove oracle signer is signaled, the action key used is:

`bytes32 actionKey = _removeOracleSignerActionKey(account);`

However, `removeOracleSignerAfterSignal` uses an incorrect action key to validate the signal. Specifically, it uses `_addOracleSignerActionKey` instead of `_removeOracleSignerActionKey`.

Therefore a signer cannot be removed using the `removeOracleSignerAfterSignal` function.

Recommendation

Use the `_removeOracleSignerActionKey` in the `removeOracleSignerAfterSignal` function.

BOU-1 | Tight Stop Loss Abuse

Category	Severity	Location	Status
Logical Error	High	BaseOrderUtils.sol: 288-298	Unresolved

Description

For stop-losses, the `triggerPrice` is used to represent the price of the asset instead of just using it as a trigger for execution. In traditional markets, the stop-loss price is not the guaranteed execution price especially in times of heavy volatility.

Users can open a long and place a SL ever so slightly below the current price. If they get stopped out then they will lose out on fees. However, with high leverage, the upside gain is immense with little risk. Such a high reward will come at the expense of the pool, hurting LPers and the market as a whole.

Ultimately, this allows sophisticated traders to have a superior strategy than that of traditional stop loss orders where they are treated like mere triggers – hurting the profitability of LPers.

Recommendation

Use the latest price (`secondaryPrice`) when executing a stop loss order rather than giving the user their exact `triggerPrice`.

ERTR-1 | UI Fee Manipulation

Category	Severity	Location	Status
Protocol Manipulation	High	ExchangeRouter.sol: 358	Unresolved

Description

The `uiFee` can be manipulated during the order/deposit/withdrawal execution to determine whether or not the action is executed and circumvent the `validateRequestCancellation` period.

Ultimately this allows malicious users to make a short-term risk-free trade as they can decide whether or not their action should be executed successfully with prices from a few blocks ago.

For example, during a withdrawal, a malicious user can re-enter the system during the execution of the first swap (WithdrawalUtils.sol: 388) into the `ExchangeRouter.setUiFeeFactor` function and change the `uiFeeFactor` for the `uiFeeReceiver` of the withdrawal executed.

The change in `uiFeeFactor` can make the difference between the subsequent swap satisfying the `minOutputAmount` — therefore deciding whether the withdrawal can go through.

Additionally, note that a similar effect can be achieved for any order/deposit/withdrawal by simply front-running the execution tx and changing the `uiFee`.

Notice that malicious `uiFeeReceivers` can manipulate the `uiFeeFactor` after a user submits their order/deposit/withdrawal. This way a `uiFeeReceiver` can promise a `uiFee` of .05%, but adjust it to be much higher right before the actual execution.

Recommendation

Do not allow the `uiFeeFactor` that is experienced in the execution to be changed after the order/deposit/withdrawal is created/updated.

ORDU-1 | Referral Code Manipulation

Category	Severity	Location	Status
Protocol Manipulation	High	OrderUtils.sol: 56	Unresolved

Description

A malicious user can manipulate the referral code associated with their account to decide whether or not a trade should be executed.

For example, the referral code could be switched to a higher discount just in time to allow a `MarketIncrease` execution tx to pass the `order.minOutputAmount()` validation and go through.

This allows a malicious trader to make a short-term risk-free trade as they can decide whether or not they should be executed successfully with prices from a few blocks ago.

Additionally, the `customDiscountShare` of a single discount code could be leveraged in the same way.

Notice that allowing the referral codes to be adjusted after orders are submitted also allows for referrer manipulations. This way referrers can promise a 2% discount but front-run order executions to adjust the `customDiscountShare` such that the trader receives no discount.

Recommendation

Store the referral discount on a per-order basis. Do not allow the referral discount to be adjusted in real-time by either the referral code being used or the `customDiscountShare` of a particular code.

BOU-2 | Stop-loss Won't Execute On Price Gap

Category	Severity	Location	Status
Logical Error	High	BaseOrderUtils.sol: 280-282	Unresolved

Description

The prices for a stop-loss are required to straddle the trigger price in `setExactOrderPrice`. In the case of a price gap where both the primary and secondary prices fall below/above the trigger price, the stop-loss will fail to execute. This will prevent a position's profit from being secured or loss to be mitigated

For example, if the trigger price is $100 for a long SL but price gaps to $99 (primary) -> $98 (secondary) the SL will not be triggered and the user will still have exposure in the market.

Recommendation

Do not revert if both primary and secondary prices fall below/above the trigger price.

DPCU-3 | Unliquidatable Position Due to PriceImpactDiff

Category	Severity	Location	Status
Logical Error	High	DecreasePositionCollateralUtils.sol: 129, 367	Unresolved

Description PoC

When a position is liquidated with `getLiquidationValues` the `pnlAmountForPool` is set to `params.position.collateralAmount() - fees.funding.fundingFeeAmount`.

However, this value does not account for the amount incremented for the claimable collateral with `incrementClaimableCollateralAmount` in the event that the price impact is capped.

This will result in a revert since the `claimableCollateralAmount` is included in the `getExpectedMinTokenBalance`. Therefore making a position unliquidatable when the price impact is capped.

Recommendation

Be sure to appropriately set aside the `collateralCache.pnlDiffAmount` in the `cache.pnlToken` when liquidations enter the `getLiquidationValues` function.

BOU-3 | Position Impact Pool Manipulation

Category	Severity	Location	Status
Protocol Manipulation	High	BaseOrderUtils.sol: 383	Unresolved

Description PoC

When calculating the PositionPricingUtils.getPriceImpactAmount the difference between the executionPrice and the latestPrice is used to derive the resulting priceImpactAmount for the positionImpactPool.

However, the executionPrice can be modified to be the acceptablePrice in the event that the acceptablePrice cannot be fulfilled by the initial max/min latestPrice. This will lead to a difference in the executionPrice and the latestPrice that is not necessarily from the priceImpactUsd amount.

Example

- Consider a LimitIncrease Long order
- priceImpactUsd is 0 for simplicity, although this applies when priceImpactUsd is nonzero
- The acceptablePrice is not fulfilled by the triggerPrice (max) so the acceptablePrice is used
- triggerPrice is used as the _latestPrice in getPriceImpactAmount
- However triggerPrice != acceptablePrice (where the acceptablePrice is my executionPrice)
- Therefore there is a nonzero priceDiff in getPriceImpactAmount, this is errantly credited as positive PI and taken out of the impact pool when there is no impact.

As a result, when the acceptablePrice is more favorable than the triggerPrice, the positionImpactPool is decreased even when the user caused a non-trivial imbalance in the OI and initially had negative priceImpactUsd.

This will influence the positionImpactPool to trend towards 0 as more orders that have an acceptablePrice that is *more favorable than* the triggerPrice are executed. Ultimately this stifles any amount of positive impact that can be offered to users to balance the OI, since the positive impact amount is capped to the balance of the positionImpactPool.

Recommendation

Consider removing the feature where the user may get their acceptable price if the first price + impact is not fulfillable and rather revert and have the order canceled/frozen if the acceptablePrice is not met.

Otherwise do not allow users to set an acceptablePrice that is *more favorable than* the triggerPrice.

MKTU-3 | Pending Borrowing Fees Brick Withdrawals

Category	Severity	Location	Status
Underflow	Medium	MarketUtils.sol: 314, 321	Unresolved

Description PoC

It is possible for user's withdrawals to revert because the pending borrowing fees are attributed to the user's withdrawal but have not yet been added to the `poolAmount`.

In cases where there is a significant amount of unpaid borrowing fees this can become a non-trivial issue for users attempting to withdraw.

Recommendation

Consider allowing a separate claiming process for borrowing fees, or implementing a pathway for regular position updates to pay the pending borrowing fees.

SWPU-1 | Max Price Used For Swap Pricing

Category	Severity	Location	Status
Logical Error	Medium	SwapUtils.sol: 188	Unresolved

Description

The `getLatestPrice` function is used to get prices for swaps, however, this will return the custom price for the token if any is set. In the case of a `MarketIncrease` long order, the min and max price for the `customPrice` are both the max of the `primaryPrice`. The inverse can be true using a `MarketDecrease` short order.

This way users can get more favorable execution while swapping for the index token during a Market order.

This invalidates the implemented protection where the `inToken` is supposedly valued at the minimum price and the `outToken` is valued at the max price:

`cache.amountOut = cache.amountIn * cache.tokenInPrice.min / cache.tokenOutPrice.max;`

Recommendation

Do not allow the max of the `primaryPrice` to be used as the price for the `tokenIn` during a swap.

SWOU-1 | LimitSwaps Unnecessarily Delayed

Category	Severity	Location	Status
Logical Error	Medium	SwapOrderUtils.sol: 67	Unresolved

Description

The `validateOracleBlockNumbers` function for `LimitSwaps` does not allow oracle block numbers to be equal to the `orderUpdatedAtBlock`. This is in contradiction to the oracle block validation for increase and decrease orders.

Additionally, this unnecessarily requires that limit swaps be executed at a delayed block number, when the current block number may provide a more favorable execution for the trader.

Recommendation

Change the requirement from `!minOracleBlockNumbers.areGreaterThan(orderUpdatedAtBlock)` to `!minOracleBlockNumbers.areGreaterThanOrEqualTo(orderUpdatedAtBlock)`.

DOU-1 | Minimum Output Amount Grieifing

Category	Severity	Location	Status
Griefing	Medium	DecreaseOrderUtils.sol	Unresolved

Description

A malicious actor can observe a user's `triggerPrice` for their stop-loss (among other order types) approaching and shift price impact in the user's market (or in the user's virtual inventory) such that their minimum output becomes invalidated and the order gets canceled.

In some cases this could cause significant grief to users who would have otherwise exited the market. A malicious actor may stand to benefit from this by holding MarketTokens and griefing traders within that market.

Recommendation

Document this behavior clearly to users. Monitor such manipulations and disincentivize them accordingly by adjusting the price impact factors as necessary.

KEY-1 | Wrong Key For Pool Adjustment

Category	Severity	Location	Status
Typo	Medium	Keys.sol: 757	Unresolved

Description

The `poolAmountAdjustmentKey` function in `Keys.sol` uses the `POOL_AMOUNT` key instead of the `POOL_AMOUNT_ADJUSTMENT` key.

There are luckily no catastrophic consequences as this is an `int` value and the `poolAmountKey` is a `uint`, however, it poses a significant risk to any future changes and would cause confusion/potential issues for those reading using the `POOL_AMOUNT_ADJUSTMENT` key.

Recommendation

Alter the `poolAmountAdjustmentKey` function to use the `POOL_AMOUNT_ADJUSTMENT` key.

IPU-1 | Price Impact Double Counted

Category	Severity	Location	Status
Double Counting	Medium	IncreasePositionUtils.sol: 151	Unresolved

Description

When increasing a position, `PositionUtils.validatePosition` is called after incrementing the OI for the position increase. However, the validation will re-compute the price impact amount based on this updated OI.

The resulting `cache.priceImpactUsd` in `isPositionLiquidatable` would be inaccurate to the actual price impact experienced. Therefore some positions may be errantly prevented from being opened with this validation.

Recommendation

Validate the position based on the previous OI, therefore accurately representing the OI delta of the order.

DPU-1 | Token Amount Added To USD Value

Category	Severity	Location	Status
Logical Error	Medium	DecreasePositionUtils.sol: 142	Unresolved

Description

The `estimatedRemainingCollateralUsd` is incremented by a token amount rather than a token amount multiplied by a price:

`estimatedRemainingCollateralUsd += params.order.initialCollateralDeltaAmount().toInt256();`

As a result, the main effect is `estimatedRemainingCollateralUsd` is much smaller than it should be and the position is more likely to get closed out in its entirety unexpectedly due to the `MIN_COLLATERAL_USD` check.

Recommendation

Multiply the `params.order.initialCollateralDeltaAmount()` by the price of the collateral token to receive a USD value before adding it with the `estimatedRemainingCollateralUsd`.

EDPU-1 | Users Are Negatively Affected By The Price Spread

Category	Severity	Location	Status
Logical Error	Medium	ExecuteDepositUtils.sol: 310, 322	Unresolved

Description

When the `positiveImpactAmount` is calculated during a deposit, the `_params.priceImpactUsd` is divided by the `tokenPrice.max` to be converted into an `outToken` amount. However the token amount is converted back to a USD amount when incrementing the `mintAmount`, `positiveImpactAmount.toUint256() * _params.tokenOutPrice.min`.

This means that users are negatively impacted by the price spread because they receive less positive impact than they otherwise would have.

Recommendation

Multiply the `positiveImpactAmount` by the `_params.tokenOutPrice.max` so that users are not negatively impacted by the price spread.

GLOBAL-1 | Liquidations When Features Disabled

Category	Severity	Location	Status
Logical Error	Medium	Global	Unresolved

Description

It is possible for a state to arise where liquidations are enabled but other order types are disabled. For example, increase orders can be disabled and a user is unable to add collateral to their position. Fees will accumulate until a user's position is liquidatable which leads to loss of funds.

Recommendation

Consider disallowing liquidations when a user is unable to adjust their order due to a feature being disabled.

OCL-1 | Lack Of Sequencer Uptime Check

Category	Severity	Location	Status
Validation	Medium	Oracle.sol: 588	Unresolved

Description

Even with the heartbeat validation, it may be prudent to validate that the Sequencer is active to prevent stale pricing.

This would avoid any scenarios where price movement triggers an update within the heartbeat duration but the new price is not reported to the L2, allowing traders to take advantage of stale pricing.

Recommendation

Validate whether the Sequencer is active or not. Furthermore, document keeper behavior in the case that the Sequencer is down.

CHAIN-1 | Hardcoded Chain ID

Category	Severity	Location	Status
Configuration	Medium	Chain.sol: 11	Unresolved

Description

In the `Chain.sol` file, `uint256 constant public ARBITRUM_CHAIN_ID = 42161;` is hardcoded.

From a comment in `Oracle.sol`, the codebase wishes to be impervious to a change in the chain ID:

`// it might be possible for the block.chainid to change due to a fork or similar`

However in the event that the Arbitrum chain ID changes, the `currentBlockNumber` and `getBlockHash` functions will not return the appropriate Arbitrum values, potentially causing drastic effects on the exchange.

Recommendation

Add a configurable chain ID for Arbitrum.

POSU-1 | Negative PnL Ignored In Sufficient Collateral Check

Category	Severity	Location	Status
Validation	Medium	PositionUtils.sol: 411	Unresolved

Description

The PnL of the remaining position is no longer accounted for during the `willPositionCollateralBeSufficient` check.

In the case where the remaining position PnL is positive, this avoids errantly counting profit towards the remaining position's collateral.

However, in the case where the remaining position PnL is negative, this check fails to consider that the remaining PnL could make the actual value backing the position significantly smaller than the `minCollateralUsdForLeverage`.

It may be prudent to consider the PnL of the remaining position, only when it is in a loss. This way the negative PnL, which would be subtracted from the collateral in the position, is taken into account.

Recommendation

Consider factoring the remaining position's PnL into the `willPositionCollateralBeSufficient` check, only when the remaining PnL is negative and would be subtracted from the collateral in any future order.

GLOBAL-2 | Block Re-org Attack

Category	Severity	Location	Status
Block Re-org	Medium	Global	Unresolved

Description

In the event of a block re-org a malicious trader may see that price has moved against them and decide to get their order canceled rather than recorded.

Consider the following scenario:

- Bob sees the keeper execute his `MarketIncrease` in block A
- The block re-org occurs over a period of 1 minute
- Bob sees the re-org is happening and sees that the execution of his order has since lost money and gets his tx to cancel the order recorded in block B which will come before block A.
- Bob can make his tx cancel his `MarketIncrease` by having it send the tokens necessary for the `MarketIncrease` order elsewhere.
- Bob's order is canceled rather than executed since it did not net him any profit.

Notice that there are likely many ways to exploit the two-step execution process during a re-org.

Recommendation

Beware of potential risks to the system during block re-orgs and communicate that risk with users. Consider implementing a mechanism to freeze all orders that were executed during a block re-org.

GLOBAL-3 | Multiple Read-only Reentrencies

Category	Severity	Location	Status
Reentrancy	Medium	Global	Unresolved

Description

There are several instances where a user may gain control over the order execution tx before the `dataStore` has been properly updated.

Firstly, in the `SwapOrderUtils.swap` function, the `SwapUtils.swap` is executed before the swap order is removed from the `dataStore`. Since the swap can have `shouldUnwrapNativeToken == true` the `order.receiver` will get called upon receiving the output of the swap.

Since native token transfers forward 200,000 gas (from the test environment) the receiver will have ample gas to potentially exploit any system building on top of GMX V2 and relying on the swap order in the `dataStore`.

Similarly in the `OrderUtils.cancelOrder` function, the `orderVault.transferOut` is executed before the order is removed from the `dataStore`.

Recommendation

In the `SwapOrderUtils.swap` function, remove the order with `OrderStoreUtils.remove` before the `SwapUtils.swap`. And in the `OrderUtils.cancelOrder` function, remove the order with `OrderStoreUtils.remove` before the `orderVault.transferOut`.

This way third parties building on top of GMX V2 cannot be exploited by the outdated order state in these instances.

GLOBAL-4 | Minimum Order Size

Category	Severity	Location	Status
Validation	Low	Global	Unresolved

Description

Similarly to the minimum position size, it may be prudent to add a minimum order size for both the `sizeDeltaUsd` and `initialCollateralDeltaAmount` in the case where they are greater than 0 to avoid potential manipulation.

Recommendation

Consider introducing a minimum `sizeDeltaUsd` and a minimum `initialCollateralDeltaAmount` that take effect when either of each field is not 0.

Additionally, it may be prudent to introduce similar minimums for deposits and withdrawals.

BOU-4 | Superfluous positionKey Variable

Category	Severity	Location	Status
Superfluous Code	Low	BaseOrderUtils.sol: 98	Unresolved

Description

The `ExecuteOrderParams` struct contains a `positionKey` variable that is never assigned nor referenced.

Recommendation

Remove the `positionKey` variable from the `ExecuteOrderParams` struct.

GLOBAL-5 | Revert Reason Unnecessarily Parsed

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

In the `_handleOrderError`, `_handleWithdrawalError` and `_handleDepositError` functions, the revert reason is parsed before it is necessary. In many cases the function will return before the parsed `string memory reason` is used.

Recommendation

Move the `ErrorUtils.getRevertMessage` call after the `ErrorUtils.revertWithCustomError` case to save gas in the event of a revert.

GLOBAL-6 | validateMarketTokenBalance After External Call

Category	Severity	Location	Status
Validation	Low	Global	Unresolved

Description

Throughout the codebase it is mentioned that the internal state changes of a market should be validated with `validateMarketTokenBalance` before handing over execution of the tx to a user.

However there are several places where the user may gain control of the tx before `validateMarketTokenBalance` has been called.

If a user has `shouldUnwrapNativeToken == true`, they will gain control over execution of the tx before the internal state changes of a market have been validated upon the swaps in withdrawals and orders.

Recommendation

Consider additionally validating the internal state changes of the market before potentially handing over control of the tx execution to an arbitrary address on swaps.

ERR-1 | Superfluous Error

Category	Severity	Location	Status
Superfluous Code	Low	Errors.sol	Unresolved

Description

The `InvalidFactor` error is never used.

Recommendation

Remove the `InvalidFactor` error.

EDPU-2 | Recomputed Value

Category	Severity	Location	Status
Optimization	Low	ExecuteDepositUtils.sol: 152-166	Unresolved

Description

The `cache.longTokenAmount * prices.longTokenPrice.midPrice` is re-computed when calculating price impact during a deposit but this value is already stored in the `cache.longTokenUsd`.

Similarly for `cache.shortTokenUsd`.

Recommendation

Use `cache.longTokenUsd` and `cache.shortTokenUsd` rather than re-computing these values.

PPU-1 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	PositionPricingUtils.sol: 52	Unresolved

Description

The NatSpec documentation for the `GetPriceImpactUsdParams` struct refers to a `longToken` and `shortToken` that are no longer there.

Recommendation

Update the NatSpec documentation for the `GetPriceImpactUsdParams` struct.

GLOBAL-7 | ERC-777 Tokens

Category	Severity	Location	Status
Warning	Low	Global	Unresolved

Description

Although the exchange does not intend to use ERC-777 tokens it should be emphasized that the system is vulnerable to them. Tokens with callbacks allow receivers to revert with an arbitrary revert string that can potentially cause a revert in the decoding process leading to a risk free trade opportunity.

Recommendation

Take care with the tokens able to be used on the exchange. Do not ever allow ERC-777 tokens to be used in the system.

PPU-2 | Misleading Comment

Category	Severity	Location	Status
Documentation	Low	PositionPricingUtils.sol: 323	Unresolved

Description

The comment purports that the `usdDelta` offset is necessary to prevent overflow, however it prevents underflow.

Recommendation

Update the comment to mention underflow instead of overflow.

GLOBAL-8 | Invalid Market Risk

Category	Severity	Location	Status
Warning	● Low	Global	Unresolved

Description

If there exists a single market in the GMX ecosystem that fails the `validateMarketTokenBalance` check on a swap or for any other reason, it can be leveraged to perform a short term risk free trade.

A malicious user can include one of these markets in their `swapPath` and decide whether the order should be able to go through by “plugging the hole” in the market that would otherwise fail the `validateMarketTokenBalance` check.

Recommendation

Monitor markets closely to observe if any have entered such a state and be careful to not introduce any such markets with admin intervention.

Additionally, consider using the `validateMarketTokenBalance` check on the markets involved in a users order, deposit, or withdrawal upon creation.

OCL-2 | Misleading Comment

Category	Severity	Location	Status
Documentation	Low	Oracle.sol: 331	Unresolved

Description

The comment in the `getLatestPrice` function mentions that the `acceptablePrice` may be used as the `customPrice` but this is not true.

Recommendation

Do not mention the `acceptablePrice` in this comment.

MKTU-4 | Prefer applyFactor

Category	Severity	Location	Status
Precision	Low	MarketUtils.sol: 1941-1942	Unresolved

Description

In the `getNextTotalBorrowing` function the `prevPositionBorrowingFactor` and `nextPositionBorrowingFactor` are applied without the use of `applyFactor`.

Recommendation

Favor the use of `applyFactor` for the `prevPositionBorrowingFactor` and `nextPositionBorrowingFactor`.

GLOBAL-9 | Superfluous Code

Category	Severity	Location	Status
Superfluous Code	Low	Global	Unresolved

Description

The `revertOracleBlockNumbersAreNotEqual` function is never utilized and therefore the `Errors.OracleBlockNumbersAreNotEqual` error is never thrown.

Therefore the entire `if` statement checking for `errorSelector == Errors.OracleBlockNumbersAreNotEqual.selector` in the `isOracleBlockNumberError` function can be removed.

Recommendation

Remove the unused error and related logic.

ORDU-2 | Duplicate Validation

Category	Severity	Location	Status
Superfluous Code	● Low	OrderUtils.sol: 130, 143	Unresolved

Description

An order is validated to be non-empty twice during the `createOrder` function. The `if` case on line 130 can be removed as it is duplicated by the `validateNonEmptyOrder` call.

Recommendation

Remove the bespoke `if` case.

KEY-2 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	Low	Keys.sol: 913	Unresolved

Description

There are two `claimableFundingAmountKey` functions, one with an `account` address parameter and one without. However the NatSpec for both of them references an `account` address parameter.

Recommendation

Remove the `account` parameter in the NatSpec for the `claimableFundingAmountKey` which does not have such a parameter.

DS-1 | Errant Import

Category	Severity	Location	Status
Superfluous Code	Low	DataStore.sol: 7	Unresolved

Description

The `Printer.sol` file is errantly imported into the `DataStore.sol` file.

Recommendation

Remove the unnecessary import.

OCL-3 | Direct Use Of block.timestamp

Category	Severity	Location	Status
Consistency	Low	Oracle.sol: 609	Unresolved

Description

Throughout the codebase `Chain.currentTimestamp` is utilized however in the `_getPriceFeedPrice` function `block.timestamp` is used directly.

Recommendation

Use `Chain.currentTimestamp` rather than `block.timestamp` directly.

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