GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

GMX

Summary

Audit Firm: Guardian Audits

Client Firm: GMX

Final Report Date - October 24th, 2022

Audit Summary

After a line by line manual analysis and automated review, Guardian has concluded that:

- GMX's smart contracts have an **ACTIVE OWNERSHIP**
- Important privileged address jurisdictions:
  - Execute any order or liquidation at any arbitrary price
  - Defer execution of any arbitrary deposit, withdrawal, order, or liquidation
  - Select the order of execution for all incoming deposits, withdrawals, and orders
  - Power to shut off any feature – including withdrawals and closing positions
  - Change out the protocol-wide variables used in the data store
- GMX's privileged addresses have numerous "write" privileges. Centralization risk correlated to the active ownership is **HIGH**

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**

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

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

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics
Commit(s)	298c6d7f1ef089a1437dc7099db1e4c647ed1b7e

Audit Summary

Delivery Date	October 24th, 2022
Audit Methodology	Static Analysis, Manual Review, Test Suite

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	9	9	0	0	0	0
High	4	4	0	0	0	0
Medium	11	11	0	0	0	0
Low	30	30	0	0	0	0

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
BNK	Bank.sol	fbc1cfe2e3c210832ee8c9daaa37f171f5a16c61
SBNK	StrictBank.sol	4ce4ad2ee277aaa9f61fce57253c4ce278011752
DAS	DataStore.sol	17cdcb7f4ade73358b71fd44cc5584a7d0f216a3
KEY	Keys.sol	1a864acada800287348b15467f4bc081479532e7
DEP	Deposit.sol	4f03bb4571019aaf1c31d84aebb40c6c5aed7256
DEPS	DepositStore.sol	e226c151980035976b285c13bed5242e90a4b77a
DEPU	DepositUtils.sol	ccf1e2c141e63ccb3cea34c579aaad377cd0d500
DEPH	DepositHandler.sol	03f1efcaf5218c5e9a6bffa374d43fa5ad4eb0ae
LIQH	LiquidationHandler.sol	714e8d9a50094e3b30e340432e401bdd8e9cfe60
ORDH	OrderHandler.sol	cb089fc8dedeeb2ecaf8262f7b2b878d561f4df8
WTDH	WithdrawalHandler.sol	aa0cf4b96b348297e3c0835bfb340da41bfffd39
FTU	FeatureUtils.sol	8278cda1d0f1526196e0b9bc8ab1f1ea769b7fb6
FEU	FeeUtils.sol	36c7e4ccac7f6de4d39713ba6265b482b1d1fc40
GSU	GasUtils.sol	3cf0e78cf02a7da6a19ab3e97e864bfbbed48f4e
GOV	Governable.sol	1de5f28b951dc6ed9873fb46376f09a16fa1300a
MKT	Market.sol	778270d091fe92699dd2f34a1188f4db96a31899
MKTF	MarketFactory.sol	64d414334e0caaeac2f806a10fe34c959f70a777
MKTS	MarketStore.sol	fbfb47f1626ee8d3c0ebb3b03b3eb76c4957dab3
MKTT	MarketToken.sol	871c54c34ddbca99ae9e3c47cc640a0aa992d8ce
MKTU	MarketUtils.sol	c45b521c4c564f1c3afbd8c117b8e66d1a87c466

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
NCU	NonceUtils.sol	b9a7c518fe4fa7876aee1879d7c207450f1379cf
PFE	IPriceFeed.sol	bdd4e330e580dfefe315d40d056a729604889e9e
OCL	Oracle.sol	24fb9c059109d566c60dc3e361f1afe13fa55e43
OCLM	OracleModule.sol	a7b0a3f5e3f77b984dab0bf8cfcadd42f2982c97
OCLS	OracleStore.sol	5d9ea2d40c2a90293d52e959d216ab787e20948c
OCLU	OracleUtils.sol	55a8f8f171457f6494cfa568558273e9c1542742
DOU	DecreaseOrderUtils.sol	44346ea602fc957029a5a9208e44473a1120e586
IOU	IncreaseOrderUtils.sol	047298ab3d3f6af679c2a710861f73fb522515a1
LIQU	LiquidationUtils.sol	df84957313f6548ba2b40afd3c762724e5a7cb08
ORD	Order.sol	2f34dc3f7e0252e43fe39d01d9dc58bd69c91c65
ORDS	OrderStore.sol	c191f24202e3808f648d8f905919378634ab12b8
ORDU	OrderUtils.sol	5a0587834c2ac4dbcd433c82178eff2eb3e74849
SWOU	SwapOrderUtils.sol	4c8b4ba67bab42fb6f7a4e3bd8c7b79734c522d5
DPU	DecreasePositionUtils.sol	ce0fe4c8a5f4163b05ca7f2481aba95ddd2b3d75
IPU	IncreasePositionUtils.sol	dfc5fe8caf7f3b8e0faa8dee5b16ad1e91bfb3a1
POS	Position.sol	2d5fb08cf15d56d453be13a6e351588bd906ef31
POSS	PositionStore.sol	6da7f57951d5848378754352b69c726759c46fc5
POSU	PositionUtils.sol	78a326eb4dcdecde44f868919ee3ed770aa25f4d
PPU	PositionPricingUtils.sol	76e8456058c9b72d6849dd8ceba8c2bf5a6eefa2

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
PRU	PricingUtils.sol	d7e5c4e2fe2223fc20a2ccd502b8154b171841b8
SPRU	SwapPricingUtils.sol	99b0052e9e5e7c46eca980a9b2ebb3a9242de953
ROL	Role.sol	abf414953f65c1e20d3d82b31813c863315dd284
ROLM	RoleModule.sol	eab30a06c97cccc7008b6dd9a5c288ee4f535173
ROLS	RoleStore.sol	6c6d19c1a9546ddbe6877fe07d8aba0ecabada11
ERTR	ExchangeRouter.sol	4f6497eef67d9526ef0ef52b5baf2e29638bc90b
RTR	Router.sol	87930daad7a64b71c5f8b03ec815278478f7ea19
SWU	SwapUtils.sol	54b3aab677d76c5a50e54580ed50419e82f95506
ARR	Array.sol	e66c23a6bb4b015307efc32fa4544a7ba4454b28
BIT	Bits.sol	dec11afd34e1abb3437881ce32b802a5a78d3f8e
CLC	Calc.sol	09a17056e8afa54e071511c9d4b1833ee85bd53c
ENM	EnumerableValues.sol	ae412817294d3aa09bc25ab85d1c5b7336a4195b
PRC	Precision.sol	fbfab41fc2b88aeb978ed4b7f2b68957e491290d
WTD	Withdrawal.sol	5041d0466286bbe8faf3a106d11a6809d997ae20
WTDS	WithdrawalStore.sol	9978955de4babe7e87ecdf4664b28431946846e6
WTDU	WithdrawalUtils.sol	a631ff3e9a32a6fd612f4b890891d8778c951655
FR	FeeReceiver.sol	2277540bb2cb1b0d354671d5a5a0d662643ad771

Audit Scope & Methodology

Vulnerability Classifications

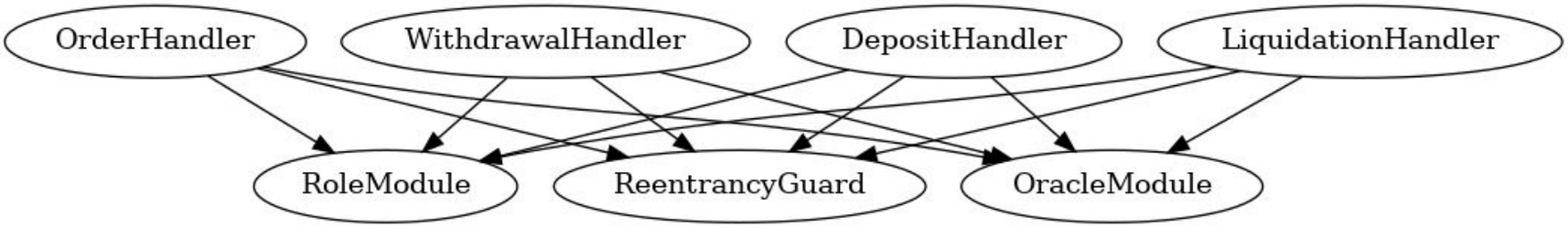
Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract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

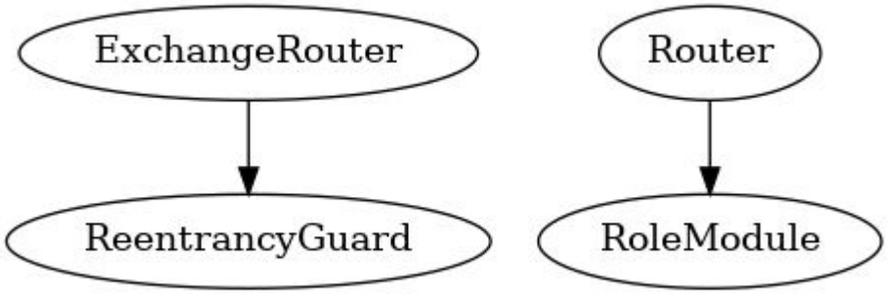
Methodology

The auditing process pays special attention to the following considerations:

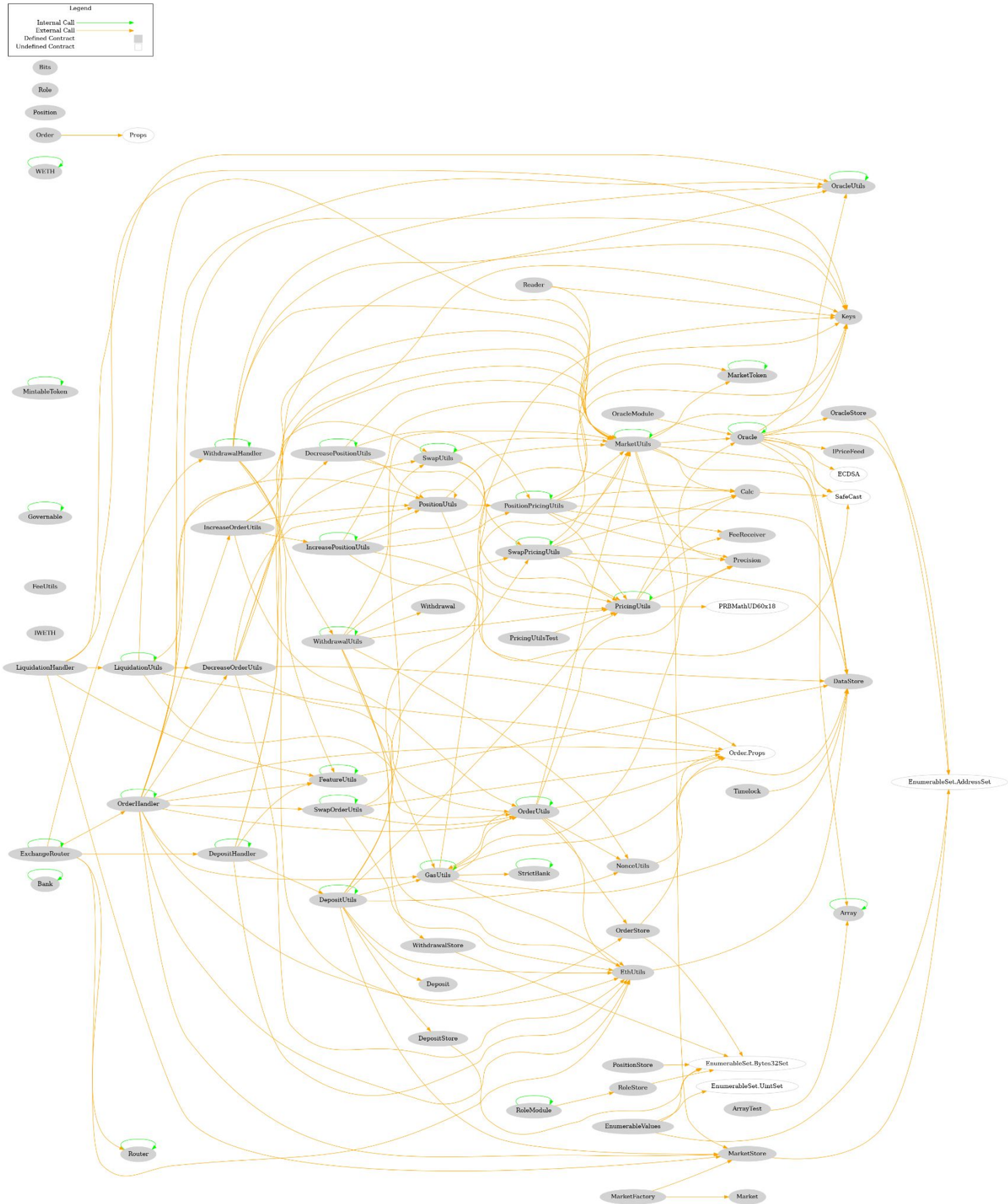
- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and 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.

Inheritance Graph





Call Graph



Findings & Resolutions

ID	Title	Category	Severity	Status
ORDU-1	Limit Increase With Same Block Pricing	Logical Error	Critical	Unresolved
GLOBAL-1	Market Used as Limit	Protocol Manipulation	Critical	Unresolved
GLOBAL-2	Delay Limit Success	Protocol Manipulation	Critical	Unresolved
DOU-1	Partial Decrease Block Not Updated	Protocol Manipulation	Critical	Unresolved
BNK-1	Bank Cannot Receive ETH	Cannot Receive Ether	Critical	Unresolved
DOU-2	Loss of Funds on Swap Path	Logical Error	Critical	Unresolved
MKTU-1	Pool Value With Inverse PnL	Logical Error	Critical	Unresolved
LIQH-1	Uninitialized Order Store	Missing Variable	Critical	Unresolved
LIQH-2	Incorrect Blocks For Liquidation	Logical Error	Critical	Unresolved
GLOBAL-3	Cannot Withdraw Backed Position	Logical Error	High	Unresolved
ORDH-1	Waste Keeper Gas	Protocol Manipulation	High	Unresolved
FR-1	Cannot Withdraw Fees	Trapped Funds	High	Unresolved
GLOBAL-4	Arbitrage Attack	Protocol Manipulation	High	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-5	Centralization Risk	Centralization / Privilege	Medium	Unresolved
OCL-1	Potential Gas DoS	Denial-of-Service	Medium	Unresolved
FR-2	Lack of Access Control For Events	Access Control	Medium	Unresolved
DEPU-1	Execution Fee DoS	Denial-of-Service	Medium	Unresolved
ERTR-1	Missing Validation	Input Validation	Medium	Unresolved
OCL-2	Chainlink Feed Validation	Pricefeed Validation	Medium	Unresolved
ORDH-2	Phantom Decrease	Logical Error	Medium	Unresolved
OCL-3	Block Number Validation	Input Validation	Medium	Unresolved
DEPU-1	Unable to Decrease Collateral	Logical Error	Medium	Unresolved
POSU-1	Cannot Liquidate 0 Remaining Collateral	Logical Error	Medium	Unresolved
MKTF-1	Same Short and Long Token	Input Validation	Medium	Unresolved
ORD-1	Typo	Typo	Low	Unresolved
OCL-4	Jagged Set of Arrays	Input Validation	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GOV-1	Pull Over Push Ownership	Ownership / Privilege	Low	Unresolved
GOV-2	Zero Address Checks	Best Practices	Low	Unresolved
ERTR-2	Zero Address Checks	Best Practices	Low	Unresolved
GLOBAL-6	Immutability Modifiers	Mutability	Low	Unresolved
ORDU-2	Unused Custom Revert	Superfluous Code	Low	Unresolved
DEPH-2	Visibility Modifiers	Visibility	Low	Unresolved
ORDH-3	Visibility Modifiers	Visibility	Low	Unresolved
SWU-1	Redeclared Variables	Optimization	Low	Unresolved
GLOBAL-8	Unclear Naming	Readability	Low	Unresolved
DEPU-2	Redundant Parameters	Superfluous Code	Low	Unresolved
GLOBAL-9	Flexible Withdrawals	Unique Behavior	Low	Unresolved
ORDH-4	Silent Cancellation	Events	Low	Unresolved
BNK-2	Duplicate Validation Checks	Superfluous Code	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-10	Lack of Events	Events	Low	Unresolved
POSS-1	Visibility Modifiers	Visibility	Low	Unresolved
GLOBAL-11	Storage Location	Optimization	Low	Unresolved
GLOBAL-12	Default Value Assignment	Optimization	Low	Unresolved
GLOBAL-13	Array Length Computation	Optimization	Low	Unresolved
GLOBAL-14	Shorten Revert Strings	Optimization	Low	Unresolved
GLOBAL-15	Division/Multiplication Optimization	Optimization	Low	Unresolved
DEPU-2	Recalculating Stored Values	Optimization	Low	Unresolved
ORDU-3	Unnecessary Variable	Optimization	Low	Unresolved
IOU-1	Validation Optimization	Optimization	Low	Unresolved
GLOBAL-16	Custom Reverts	Optimization	Low	Unresolved
WTDU-1	Unnecessary Validation	Optimization	Low	Unresolved
DEPU-3	Unnecessary Variable Set	Superfluous Code	Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-17	For-loop Increment	Optimization	Low	Unresolved
GLOBAL-18	Visibility Modifiers	Optimization	Low	Unresolved

ORDU-1 | Limit Increase With Same Block Pricing

Category	Severity	Location	Status
Logical Error	● Critical	OrderUtils.sol: 295, 306	Unresolved

Description - PoC

In `validateOracleBlockNumbersForPosition`, both conditionals on lines 295 and 306 handle the `LimitIncrease` order type but with different oracle block requirements.

During execution, the first `if` statement is entered, therefore a `LimitIncrease` order is forced to use prices where the `oracleBlockNumbers` are equal to the `orderUpdatedAtBlock`. This contradicts the behavior outlined in the Order.sol documentation where:

“LimitSwap and LimitIncrease orders can always be executed if the right prices are reached due to this, validating that the prices presented are for blocks **after the order's updatedAtBlock** should be sufficient to prevent gaming of the pricing”.

Because `LimitIncrease` orders can only be executed with prices from their `updatedAtBlock` they are rendered nearly useless.

Recommendation

Remove `LimitIncrease` from the first conditional, as it is currently duplicated.

Resolution

GLOBAL-1 | Market Used as Limit

Category	Severity	Location	Status
Protocol Manipulation	● Critical	Global	Unresolved

Description - PoC

A `MarketIncrease` order can mimic a limit order by specifying a `minOutputAmount` to fabricate a limit price. However, `MarketIncrease` orders will always be executed at the current market price, rather than limit and stop loss orders which always execute at the `acceptablePrice`. This behavior allows traders to obtain an unfair advantage as they can now have a market order acting as a traditional limit order (execution price <= `acceptablePrice`) rather than simply using a limit order (execution price == `acceptablePrice`). Combined with a stop loss, risk-free trades could be made.

Consider the following scenario:

- The current price of ETH is $2,000
- User A creates a stop-loss order for ETH at $1,800
- User A creates a market order for ETH with a `minOutputAmount` such that the market order is only executable when ETH is at or below the price of $1,800.
- The price of ETH drops to $1,700
- The market increase order is then executed at $1,700 while the stop-loss order is executed at the guaranteed $1,800
- User A is able to profit $100/ETH risk free.

Recommendation

Consider cancelling the `MarketIncrease` order if the `minOutputAmount` is not met. Alternatively, standardize the behavior of limit and stop loss orders so that they behave similarly to a `MarketIncrease` order with a `minOutputAmount`, taking the current price when it is `<=` the `acceptablePrice` rather than taking the `acceptablePrice`. However, protocol gaming would have to be taken into account.

GLOBAL-2 | Delay Limit Success

Category	Severity	Location	Status
Protocol Manipulation	● Critical	Global	Unresolved

Description - PoC1 PoC2

The try/catch block in each handler does not catch any errors related to a Panic exception or custom revert. This enables a number of bugs and exploits on the exchange.

Most notably, It is possible to create a `LimitIncrease` order that exceeds the max reserved usd, therefore reverting on execution but remaining in the order store, then when conditions are favorable the user can deposit enough reserves so their order gets executed.

Consider the following scenario:

- The current price of ETH is $2,000
- User A creates a `LimitIncrease` long order for ETH at $2,000 in block 100 with usd size larger than allowable
- User A's order reverts due to insufficient reserves but remains in the order store
- 20 blocks later ether is now $2,100
- User A deposits enough reserves to execute their `LimitIncrease`
- The `LimitIncrease` long order now executes with the prices from block 101
- User A closes their position making risk free profit

Recommendation

Utilize `catch (bytes memory lowLevelData)` to catch Panic exceptions and custom reverts.

DOU-1 | Partial Decrease Block Not Updated

Category	Severity	Location	Status
Protocol Manipulation	● Critical	DecreaseOrderUtils.sol: 46	Unresolved

Description - PoC

Decrease orders that are partially filled are not removed from the orderStore and continue to exist in perpetuity. However the amount the phantom decrease position order is left with is equivalent to the amount your position was just decreased by *not* the amount that is leftover from the original decrease position order.

Additionally, because the order is not touched upon a partial decrease order, it is possible for a user to continually submit increase orders and realize immediate and risk-free profits.

Consider the following scenario:

- The current price of ETH is $2,000
- User A creates a MarketIncrease long order for ETH at $2000 with $1,000,000 sizeInUSD
- User A's position is opened with a size of $1,000,000, and increases to a size of $1,050,000 when ETH reaches a price of $2,100
- User A sends a MarketDecrease order with sizeDeltaUSD $1,050,001 when ETH is $2,100
- User A's position is closed but the decrease order remains in the store for $1,050,000
- The phantom MarketDecrease is not touched, therefore it must be executed at $2,100
- User A submits a MarketIncrease long order when ETH is back at $2,000 for $1,049,999
- The newly created position is closed by the MarketDecrease order at $2,100 for immediate profit
- The MarketDecrease order continues to exist in the orderStore, and as long as the price of ETH is below the decrease order price, the user can keep submitting large longs and realizing profit

Recommendation

Do not allow partially fulfilled orders to remain in the orderStore, or fix the partial fulfillment logic and address the block number at which these partially fulfilled orders can be executed at.

BNK-1 | Bank Cannot Receive ETH

Category	Severity	Location	Status
Cannot Receive Ether	● Critical	Bank.sol	Unresolved

Description - PoC

Bank.sol has no receive/fallback function defined to be able to accept ether, therefore it is impossible for the Bank to withdraw ether from the WETH contract and by extension it is impossible to transfer ether out of the Bank.sol contract.

Because of this:

- Swap orders with `hasCollateralInEth == true` lead to complete loss of funds for users.
- Orders, deposits, and withdrawals using `hasCollateralInEth == true` are unable to be canceled. This means users are unable to cancel `hasCollateralInEth` orders and the keeper is unable to cancel failing orders, deposits, and withdrawals leading to phantom entries in each corresponding store.
- `StopLoss` orders with no `swapPath` and `hasCollateralInEth == true` cannot be executed as they attempt to pay the user in ether from the bank.
- Additionally, users are unable to use the `hasCollateralInEth` flag to receive ether when withdrawing or exercising a decrease order.

Recommendation

Implement a receive function in the Bank.sol contract.

Resolution

DOU-2 | Loss of Funds on Swap Path

Category	Severity	Location	Status
Logical Error	● Critical	DecreaseOrderUtils.sol: 63	Unresolved

Description - PoC

For decrease orders with a defined `swapPath` the `inputAmount` for the swap is the `initialCollateralDeltaAmount`, which cannot be set for decrease order types and defaults to 0.

Therefore, if a user supplies a `swapPath` on a decrease order, they will experience a complete loss of funds. Zero tokens would be swapped and sent back to the user, yet their position would still decrease/close.

Recommendation

Provide the correct `inputAmount` for the swap that corresponds to how much the user's position was decreased by.

Resolution

MKTU-1 | Pool Value With Inverse PnL

Category	Severity	Location	Status
Logical Error	● Critical	MarketUtils.sol: 161	Unresolved

Description - PoC

The pool value interprets traders profiting as an increase of value for the pool, and traders losing as a decrease of value of the pool. This is exactly the inverse of what should be happening. When traders lose and their loss is realized, their collateral is taken into the pool and the value of the pool increases.

Recommendation

Use `return Calc.sum(value, -pnl)`.

Resolution

LIQH-1 | Uninitialized Order Store

Category	Severity	Location	Status
Missing Variable	● Critical	LiquidationHandler.sol	Unresolved

Description

The liquidation handler has no reference to the `orderStore`, and does not assign one in the params to `processLiquidation`. Therefore, liquidations revert when attempting to access the `orderstore` on the `ExecuteOrderParams` params. Therefore it is impossible to perform liquidations on any position.

Recommendation

Include the `orderStore` in the `ExecuteOrderParams` in the `liquidatePosition` function.

Resolution

LIQH-2 | Incorrect Blocks For Liquidation

Category	Severity	Location	Status
Logical Error	● Critical	LiquidationHandler.sol	Unresolved

Description

In `processLiquidation`, the `marketDecrease` order that is created for liquidation does not get touched, therefore the `updatedAtBlock` is always 0.

Therefore it is impossible to execute a liquidation with the correct block number for oracle prices and by extension impossible to liquidate any position.

Furthermore, the `oracleBlockNumber` requirements for a `MarketDecrease` order stipulate that the prices come from the block in which the order was updated at. This would demand that the oracle provide finalized prices for the block in which the liquidation transaction is to be executed, which is impossible as the block has not yet been confirmed.

Recommendation

Resolve the contradiction in block numbers for liquidations and `MarketDecrease` orders. Additionally, make sure to call `touch()` on the order when creating it in the `processLiquidation` function.

Resolution

GLOBAL-3 | Cannot Withdraw Backed Position

Category	Severity	Location	Status
Logical Error	High	Global	Unresolved

Description - PoC

If a user creates a long position with the short token as collateral, they may only withdraw the long token or short token if there is enough short token liquidity in the pool. This can lead to circumstances where users are unable to withdraw from their positions even though they are technically backed.

This is because positions are keyed based on the `initialCollateralToken` and decrease orders must specify the same `initialCollateralToken` so there must exist enough short token liquidity to pay out the position for a decrease order.

This is strange as long positions are backed by long tokens, thus the user should be able to withdraw the long token upon decrease without relying on adequate short token liquidity to perform a swap.

Recommendation

Allow users to withdraw the long token in this case, or prevent such a scenario from happening in the first place in the `validateReserves` function.

Resolution

ORDH-1 | Waste Keeper Gas

Category	Severity	Location	Status
Protocol Manipulation	High	OrderHandler.sol: 176	Unresolved

Description - PoC

An order's executionFee is validated when the keeper is already attempting to execute the order. A malicious user can send a large number of orders that are unable to be executed and have an executionFee of 0.

This will waste the keeper's gas at no cost to the user each time execution is attempted, potentially delaying the execution of other deposits, withdrawals, and orders on the exchange.

Recommendation

Validate the executionFee upon order creation, similarly to deposits.

Resolution

FR-1 | Cannot Withdraw Fees

Category	Severity	Location	Status
Trapped Funds	● High	FeeReceiver.sol	Unresolved

Description

The `FeeReceiver` contract is responsible for receiving fees for the entire protocol, but provides no functions to retrieve erc20 tokens nor ether. Therefore any fees sent to the `FeeReceiver` contract are lost.

Recommendation

Add a withdraw function with proper access controls to withdraw and manage fees.

Resolution

GLOBAL-4 | Arbitrage Attack

Category	Severity	Location	Status
Protocol Manipulation	High	Global	Unresolved

Description

Documentation in README.md suggests that spot prices from multiple exchanges will be used to determine prices for execution. Such a price collection scheme potentially allows for economically viable price manipulation/arbitrage attacks. Attackers may be able to manipulate prices to game orders into guaranteed profits, or cause mass liquidations.

Recommendation

Adopt a TWAP with multiple price readings to make such attacks economically unviable. Otherwise, be prepared to use failsafes such as open interest caps to limit these attacks.

Optionally, order book depth/liquidity for each exchange should be considered in the calculation to further limit the scope of manipulation attacks.

Resolution

GLOBAL-5 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Medium	Global	Unresolved

Description

The exchange keeper and other permissioned addresses have the power to do nearly anything:

- Execute any order or liquidation at any arbitrary price
- Defer execution of any arbitrary deposit, withdrawal, order, or liquidation
- Select the order of execution for all incoming deposits, withdrawals, and orders
- Power to shut off any feature – including withdrawals and closing positions
- Change out the protocol-wide variables used in the `dataStore`

There are many assumptions made on the form of the price input from the keeper. Adding validation for the expected format and range of acceptable inputs would reduce the risk of high-cost mistakes, and assist in limiting the scope of the internal exploits available to the keeper.

The keeper must diligently execute orders, for example stop loss orders must be executed with the correct range of block numbers while staying within the `MAX_ORACLE_BLOCK_AGE` limit.

Recommendation

Treat the keeper's private key(s) with the utmost level of security and introduce as many safeguard checks as possible to limit the scope of the keepers potential attack vectors.

Resolution

OCL-1 | Potential Gas DoS

Category	Severity	Location	Status
Denial-of-Service	Medium	Oracle.sol: 82	Unresolved

Description

There is a potential DoS with the `setPrices` function, as the lengths of the `tokens`, `signers`, `compactedOracleBlockNumbers`, and `compactedPrices` arrays are unbounded. The `for` loop through these arrays in addition to memory expansion costs may result in gas fees exceeding the block gas limit, especially with long swap routes that require many token prices.

Recommendation

Carefully consider the max signer bounds to limit gas usage. Furthermore, consider bounds on the number of tokens, as all signers are iterated over for each token.

Resolution

FR-2 | Lack of Access Control For Events

Category	Severity	Location	Status
Access Control	Medium	FeeReceiver.sol: 8	Unresolved

Description

Anyone can call `notifyFeeReceived` as there is a lack of access control, which leads to a depreciation of the authenticity of the event.

Recommendation

Implement access control concerning who may call `notifyFeeReceived`.

Resolution

DEPU-1 | Execution Fee DoS

Category	Severity	Location	Status
Denial-of-Service	Medium	DepositUtils.sol: 74	Unresolved

Description

There is a potential DoS with `createDeposit`, as a malicious address may send a miniscule amount of WETH to the deposit store such that another user's WETH deposit no longer matches the `executionFee` in their deposit. Thus, the deposit execution reverts and is unable to be created.

Recommendation

Consider removing the strict equality for the `executionFee` or handling the accounting in the `depositStore` such that another address cannot increase the deposit amount for a user.

Resolution

ERTR-1 | Missing Validation

Category	Severity	Location	Status
Input Validation	Medium	ExchangeRouter.sol: 50, 106	Unresolved

Description

When creating a deposit, there is no validation that the `longToken` or the `shortToken` are valid for the supplied market. If WBTC is accidentally used as the long token in an ETH/USDC market, the user would lose their WBTC in the `depositStore`.

Furthermore, there is no validation that either the long token amount or short token amount is non-zero. This check should be added to prevent the keeper from executing trivial deposits.

Additionally, when creating an order, there is no validation that the initial collateral token is valid for the provided market, which can lead to invalid orders stored in the `orderStore`.

Recommendation

Implement the above mentioned validations.

Resolution

OCL-2 | Chainlink Feed Validation

Category	Severity	Location	Status
Pricefeed Validation	Medium	Oracle.sol: 300	Unresolved

Description

Extra validation checks should be added on the result from the Chainlink price feed to ensure non-stale data. The price from the data feed influences the execution of orders and liquidations so it is imperative the data is up to date and correct.

Recommendation

Add the following require statements to validate the price feed:

- `require(answeredInRound >= roundID, "Chainlink:: Stale price")`
- `require(timestamp > 0, "Chainlink:: Round not complete")`

Resolution

ORDH-2 | Phantom Decrease

Category	Severity	Location	Status
Logical Error	Medium	OrderHandler.sol: 88	Unresolved

Description

It is possible to have a decrease order in the `orderStore` when no corresponding position is present because orders are not cancelled upon failure with the `EMPTY_POSITION_ERROR_KEY`.

Consider the following scenario:

- User A creates a position
- User A sends a `LimitDecreaseOrder` and a `MarketDecreaseOrder` in the same block
- `LimitDecreaseOrder` gets executed first
- `MarketDecreaseOrder` is now floating on an empty position and it never gets canceled
- User A opens a new position with an increase order of the same market, collateral token, and directionality
- The market decrease order unexpectedly affects User A's new position as it was waiting to be executed in the `orderStore`

Recommendation

Cancel the order upon failed execution in the case of `EMPTY_POSITION_ERROR_KEY`.

Resolution

OCL-3 | Block Number Validation

Category	Severity	Location	Status
Input Validation	Medium	Oracle.sol: 229	Unresolved

Description

According to documentation, the price for a token is only retrieved once a block is finalized.

However, `_setPrices` does not revert when the block number of the prices provided by the oracle equals the current block number: `if (cache.oracleBlockNumber > block.number) { revert … }`.

This allows the keeper to provide prices for a block number that is equivalent to the current block and waste gas when it would likely continue on to revert when validating signatures.

Recommendation

Use `>=` instead of `>`.

Resolution

DPU-1 | Unable to Decrease Collateral

Category	Severity	Location	Status
Logical Error	Medium	DecreasePositionUtils.sol: 157	Unresolved

Description

In the `processCollateral` function, the `collateralDeltaAmount` is initially set to `params.order.initialCollateralDeltaAmount().toInt256()`, which cannot be set for decrease orders. Therefore, there is no way to decrease collateral from a position without closing the entire position.

Recommendation

Allow for `initialCollateralDeltaAmount` to be set for decrease orders so users are able to decrease their collateral without closing their position.

Resolution

POSU-1 | Cannot Liquidate 0 Remaining Collateral

Category	Severity	Location	Status
Logical Error	Medium	PositionUtils.sol: 154	Unresolved

Description

There is potential for a position with $0 of `remainingCollateral` to avoid liquidation. This is due to the fact that the first conditional can avoid being triggered if the `MIN_COLLATERAL_USD` was not set. If so, the second conditional would revert due to division by 0. Thus, a position that is liquidatable cannot be liquidated.

Recommendation

Add zero checks for such a scenario.

`if (remainingCollateralUsd == 0) return true;`

Resolution

MKTF-1 | Same Short and Long Token

Category	Severity	Location	Status
Input Validation	Medium	MarketFactory.sol: 19	Unresolved

Description

When creating a new market, nothing prevents the long token from being set to the same address as the short token.

In such a case, deposits would double count the funds you are sending to the pool. Since the creation of markets will eventually be unpermissioned, this validation is paramount.

Recommendation

Validate that the long token is not equal to the short token.

Resolution

ORD-1 | Typo

Category	Severity	Location	Status
Typo	● Low	Order.sol: 58	Unresolved

Description

On line 58, “current” is misspelled as “curent“.

Recommendation

Replace “curent” with “current” in the comment.

Resolution

OCL-4 | Jagged Set of Arrays

Category	Severity	Location	Status
Input Validation	● Low	Oracle.sol: 226	Unresolved

Description

In the `_setPrices` function, nothing prevents the `signatures` array from being a different length than the `signers` array. If there are less signatures than signers, that would lead to an out-of-bounds error and the index price would not be able to get set.

In this case it would save gas to add a length validation check before commencing potentially expensive price computations.

Recommendation

Check that the length of the `signatures` array is a multiple of the length of the `signers` array.

Resolution

GOV-1 | Pull Over Push Ownership

Category	Severity	Location	Status
Ownership / Privilege	● Low	Governable.sol: 23	Unresolved

Description

The ownership transfer process should have a push and pull step rather than executing in a single transaction with `setGov`.

This way the protocol may avoid catastrophic errors such as setting the wrong governance address.

Recommendation

Implement a two step transfer process for the `gov` role where the new `gov` address must explicitly accept its new role.

Resolution

GOV-2 | Zero Address Checks

Category	Severity	Location	Status
Best Practices	● Low	Governable.sol: 23	Unresolved

Description

In the `setGov` function there are no zero address checks on the new `gov` address.

Recommendation

Add zero address checks as an errantly set `gov` address would be catastrophic for the protocol.

Resolution

ERTR-2 | Zero Address Checks

Category	Severity	Location	Status
Best Practices	● Low	ExchangeRouter.sol	Unresolved

Description

The `createDeposit`, `createWithdrawal`, and `createOrder` functions compose the user's interface for the exchange but perform no zero address checks on the addresses they accept.

Recommendation

Add zero address checks to avoid any unexpected behavior when users interact with these functions. Document where zero addresses are expected, and enforce when they are not.

Resolution

GLOBAL-6 | Immutability Modifiers

Category	Severity	Location	Status
Mutability	● Low	Global	Unresolved

Description

Throughout the contracts many contract level variables can be declared `immutable`. Common variables such as the `dataStore`, `depositHandler`, `withdrawalHandler`, `orderHandler`, `depositStore`, `withdrawalStore`, and `orderStore` among others should be declared `immutable` in multiple contracts.

Recommendation

Implement the above recommended immutability modifiers.

Resolution

ORDU-2 | Unused Custom Revert

Category	Severity	Location	Status
Superfluous Code	Low	OrderUtils.sol: 264	Unresolved

Description

In the `setExactOrderPrice` function, the revert statement on line 264 can be replaced with a `revertUnsupportedOrderType` function call.

Recommendation

Implement the above function call.

Resolution

DEPH-2 | Visibility Modifiers

Category	Severity	Location	Status
Visibility	● Low	DepositHandler.sol: 105	Unresolved

Description

The `_executeDeposit` function is called from within the context of a try/catch, therefore it can be declared `external`.

Recommendation

Declare the function `external`.

Resolution

ORDH-3 | Visibility Modifiers

Category	Severity	Location	Status
Visibility	Low	OrderHandler: 158	Unresolved

Description

The `_executeOrder` function is called from within the context of a try/catch, therefore it can be declared `external`.

Recommendation

Declare the function `external`.

Resolution

SWU-1 | Redeclared Variables

Category	Severity	Location	Status
Optimization	Low	SwapUtils.sol: 45, 46, 47, 54	Unresolved

Description

The `market`, `nextIndex`, `receiver`, and `_params` variables can be declared outside of the `for` loop to save gas.

Recommendation

Declare these variables outside the `for` loop.

Resolution

GLOBAL-8 | Unclear Naming

Category	Severity	Location	Status
Readability	● Low	Global	Unresolved

Description

`hasCollateralInEth` is used in many different ways throughout the codebase, it should have a more adaptable/explicit name.

Recommendation

Rename `hasCollateralInEth` or assign specific names to use-cases such as deposits and withdrawals.

Resolution

DPU-2 | Redundant Parameters

Category	Severity	Location	Status
Superfluous Code	Low	DecreasePositionUtils: 146	Unresolved

Description

The `processCollateral` function accepts a `position` parameter, but the same position is already available on the `params` parameter of type `DecreasePositionParams`.

Recommendation

Remove the `position` parameter as it is available on the `DecreasePositionParams`.

Resolution

GLOBAL-9 | Flexible Withdrawals

Category	Severity	Location	Status
Unique Behavior	● Low	Global	Unresolved

Description

It should be noted that when withdrawing `marketTokens`, a user will not always be able to redeem the corresponding amounts of underlying assets that they provided. This is because other depositors may withdraw whichever long or short collateral token they wish, as long as their withdrawal matches the `marketToken` value they received for depositing into the market.

Example:

- The ETH/USD market is empty
- User A deposits $500 worth of ETH and $500 worth of USDC
- User B deposits $500 worth of ETH and $500 worth of USDC
- User A withdraws $1000 worth of USDC
- User B attempts to withdraw $500 worth of ETH and $500 worth of USDC, but the tx reverts because there is no USDC left in the pool

Recommendation

Ensure the withdrawal behavior is clearly documented and communicated to users.

Resolution

ORDH-4 | Silent Cancelation

Category	Severity	Location	Status
Events	Low	OrderHandler: 95	Unresolved

Description

When an order is executed and reverts with an error that is not retried, it gets canceled silently. There are no events emitted nor is there a return value for the `executeOrder` function to communicate whether an order was canceled or not.

Recommendation

Implement a return value or event that indicates whether or not an order was canceled.

Resolution

BNK-2 | Duplicate Validation Checks

Category	Severity	Location	Status
Superfluous Code	● Low	Bank.sol: 35, 43	Unresolved

Description

In Bank.sol the `require(receiver != address(this), "Bank: invalid receiver")` check can be deduplicated into a modifier that also validates that the receiver is a non-zero address.

Additionally, these checks can be made in `transferOut` and validate that the amount is greater than zero.

Recommendation

Implement the above refactors.

Resolution

GLOBAL-10 | Lack Of Events

Category	Severity	Location	Status
Events	Low	Global	Unresolved

Description

Throughout the contracts there are missing events for important actions such as order execution, position size modifications, liquidations, deposits, and withdrawals.

Recommendation

Implement events throughout the contracts for improved monitoring, readability, and transparency.

Resolution

POSS-1 | Visibility Modifiers

Category	Severity	Location	Status
Visibility	Low	PositionStore.sol: 50	Unresolved

Description

The function `contains` is declared as `public` but is never called from within the contract.

Recommendation

Modify the visibility from `public` to `external`.

Resolution

GLOBAL-11 | Storage Location

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

The following parameters are marked as `memory` when they could be declared `calldata`:

- PositionStore::18
- DepositHandler::73,102
- OrderHandler::58,73,155
- OrderUtils::73
- DepositUtils::61
- DepositStore::17
- WithdrawalUtils::74
- WithdrawalStore::17
- WithdrawalHandler::84,112
- OracleModule::15
- Oracle::82
- LiquidationHandler::51

Recommendation

If it is desired to reduce gas for these functions, potentially at the expense of deployment costs, change the suggested memory modifiers from `memory` to `calldata`.

Resolution

GLOBAL-12 | Default Value Assignment

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the contracts `uint` variables are initialized to the default value (0):

- MarketUtils.sol::619
- Oracle.sol::100,136,226,252,285
- OracleUtils.sol::45
- SwapUtils.sol::44
- Timelock.sol::38,44
- Array.sol::33,41,51

Recommendation

Remove the unnecessary assignments.

Resolution

GLOBAL-13 | Array Length Computations

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

Throughout the contracts array lengths should be computed outside of `for` loops:

- MarketUtils.sol::619
- Oracle.sol::22,100,252,285
- SwapUtils.sol::44
- Timelock.sol::38,44
- Array.sol::33,41,51

Recommendation

Compute array lengths outside of `for` loops.

Resolution

GLOBAL-14 | Shorten Revert Strings

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the contract, revert strings can be shortened to 32 bytes to decrease gas costs:

- WithdrawalHandler.sol::46
- IncreaseOrderUtils.sol::48
- OrderUtils.sol::264
- DecreasePositionUtils.sol::141
- WithdrawalUtils.sol::76,106,204

Recommendation

Shorten revert strings to 32 bytes.

Resolution

GLOBAL-15 | Division/Multiplication Optimization

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

Throughout the contract, switching from division/multiplication to right-shift/left-shift can save gas:

- PricingUtils.sol::64
- Array.sol::62,65

Recommendation

Switch uses of division to right-shift and uses of multiplication to left-shift.

Resolution

DEPU-2 | Recalculating Stored Values

Category	Severity	Location	Status
Optimization	Low	DepositUtils.sol:126, 127	Unresolved

Description

In the call to `SwapPricingUtils.GetSwapPricingParams`, the values stored in `longTokenUsd` and `shortTokenUsd` on lines 113 and 114 are recalculated on lines 126 and 127.

Recommendation

Use the previously computed values instead of recalculating.

Resolution

ORDU-3 | Unnecessary Variable

Category	Severity	Location	Status
Optimization	Low	OrderUtils.sol: 208	Unresolved

Description

Declaring the `price` variable on line 208 is unnecessary, as the calculated value is only used once.

Recommendation

Remove the `price` variable declaration and use the result of the calculation directly.

Resolution

IOU-1 | Validation Optimization

Category	Severity	Location	Status
Optimization	Low	IncreaseOrderUtils.sol: 12	Unresolved

Description

Validation for the `market` can occur before any transfer logic in order to save gas.

Recommendation

Move the `MarketUtils.validateNonEmptyMarket` call to the first line in `processOrder`.

Resolution

GLOBAL-16 | Custom Reverts

Category	Severity	Location	Status
Optimization	● Low	Global	Unresolved

Description

Throughout the contracts, several `require` statements can be converted to custom reverts. The following files contain statements that can be replaced:

- OrderUtils.sol::90
- WithdrawalUtils::76,204
- DepositHandler::44
- OrderHandler::53,130
- withdrawalHandler::46
- RoleModule.sol::31,36,41,46,51

Recommendation

Replace `require` statements with custom reverts.

Resolution

WTDU-1 | Unnecessary Validation

Category	Severity	Location	Status
Optimization	Low	WithdrawalUtils.sol: 288	Unresolved

Description

`MarketToken.burn` will revert on its own if the user has insufficient market tokens.

Recommendation

Remove the validation check if a custom revert is not necessary.

Resolution

DPU-3 | Unnecessary Variable Set

Category	Severity	Location	Status
Superfluous Code	Low	DecreasePositionUtils.sol: 97	Unresolved

Description

There is no need to set the position's `collateralAmount` to 0, as the position is immediately removed on line 99.

Recommendation

Remove the line setting the position's `collateralAmount`.

Resolution

GLOBAL-17 | For-Loop Increment

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the contracts numerous `for` loops have upper bounds with no risk of overflow for the index. Therefore an unchecked block can be used in the loop to increment the index and save gas.

Recommendation

Implement the above suggestion.

Resolution

GLOBAL-18 | Visibility Modifiers

Category	Severity	Location	Status
Optimization	Low	Global	Unresolved

Description

Throughout the codebase there are numerous library functions that can be declared `internal` to save gas.

Recommendation

Declare library functions that do not need to be called in an `external` context as `internal`.

Resolution

Auditor’s Verdict

After a line by line manual analysis and automated review, Guardian has concluded that:

- GMX’s smart contracts have an **ACTIVE OWNERSHIP**
- Important privileged address jurisdictions:
  - Execute any order or liquidation at any arbitrary price
  - Defer execution of any arbitrary deposit, withdrawal, order, or liquidation
  - Select the order of execution for all incoming deposits, withdrawals, and orders
  - Power to shut off any feature – including withdrawals and closing positions
  - Change out the protocol-wide variables used in the data store
- GMX’s privileged addresses have numerous “write” privileges. Centralization risk correlated to the active ownership is **HIGH**

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