GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

GMX

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand

Client Firm GMX

Final Report Date September 15th, 2023

Audit Summary

GMX engaged Guardian to review the security of its dynamic funding fees and position impact distribution updates for the GMX V2 system. From the 1st of September to the 15th of September, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

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

🔗 Blockchain network: **Arbitrum, Avalanche**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/GMX-Updates-9-4-23

Table of Contents

Project Overview

Project Summary

Project Name	GMX
Language	Solidity
Codebase	https://github.com/gmx-io/gmx-synthetics/
Commit(s)	d9e14200599ebbc11963cfe5ab7d68d416a600c7

Audit Summary

Delivery Date	September 15th, 2023
Audit Methodology	Static Analysis, Manual Review, Test Suite

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	1	0	0	0	0
High	1	1	0	0	0	0
Medium	10	10	0	0	0	0
Low	10	10	0	0	0	0

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
CON	Config.sol	1a4145822d9b782da1ca5ab00e9c1cc52d6b0f3e
TIME	Timelock.sol	b736137c020faa12784ac2dc196e7f40023ba3fb
KEYS	Keys.sol	dfbf253947bb0efe3a3b7df668ce492bcc09a0fb
DEPU	DepositUtils.sol	fcfb6622a110c1909a9e3caa7051cae2e0f09cdd
EDPU	ExecuteDepositUtils.sol	709b75d01cc2ee760984ba1802e64ac81d185032
ERR	Errors.sol	01d8b5f1f39479244515754145c5d6edb2edd736
ADLH	AdlHandler.sol	5473b3239e77c1c1b910b17ed67ebcccae7f0313
BOH	BaseOrderHandler.sol	ab0e9a8826878341190 45de9af1ed7888a150bad
DEPH	DepositHandler.sol	f1b26153aa03ce69127f7c09d9d1c8760ca93a7b
LIQH	LiquidationHandler.sol	c6a811998f32537ebce9efebd5adb3def4b51ca9
ORDH	OrderHandler.sol	595a191cad82539de436bc4c527f01ef11e2128a
WTDH	WithdrawalHandler.sol	d0c8318da5f911f0a00bc8e86d7e7da664ee001d
GSU	GasUtils.sol	3361e8a1ba7c1585691b55bfc30e69018b9917e0
MKTU	MarketUtils.sol	e2043d1eb587aaebe238b3dd766a1ac8fb300037
OCL	Oracle.sol	72a18d62d7161dcb9df05b18dc4604e8c8e2afa8
OCLU	OracleUtils.sol	07fc421e8f8b8d33c861e72d272d7937d01eb29d
ORDU	OrderUtils.sol	57857ff24dfd6b5fa7a79a89a502c44c71b7b4e4
DPCU	DecreasePositionCollateralUtil.sol	e22b1f528f408a5519d5897a41a031a9dd8460b0
DPU	DecreasePositionUtils.sol	23ec322a04cdb8eea423c86961f149e0e37cbe95
IPU	IncreasePositionUtils.sol	9cd4abe69ba22de4b02f039bb5c720535c635f4f

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
POSU	PositionUtils.sol	b10e975d1b41120f3c0ec42c0652ce11e2e79c0a
PPU	PositionPricingUtils.sol	6537ac9e79efba2618ac44b42c7e7a2595cfc2e5
SWPU	SwapPricingUtils.sol	9207c92d77cbb931e16c4b8afbe73f6af1a29cbd
REFU	ReferralUtils.sol	97a55ae27e323b808665b755b42012b08da4cedd
CALC	Calc.sol	bd7fca9ee4a784b2a8d6089baffa7f14ff08f996
EWDU	ExecuteWithdrawalUtils.sol	a88b3932dcf902324e6f4ba1b68d1a1283d2ec11
WTDU	WithdrawalUtils.sol	f58780b028ddbdd2e621c5f52fac5cc6fa07c5cc

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.

Findings & Resolutions

ID	Title	Category	Severity	Status
CALC-1	boundMagnitude Fails To Bound Magnitude	Logical Error	Critical	Pending
MKTU-1	Funding Factor Per Second Open To Manipulation	Protocol Manipulation	High	Pending
CALC-2	boundMagnitude Function Cannot Bound 0	Logical Error	Medium	Pending
MKTU-2	minFundingFactorPerSecond != 0 Undesired behaviors	Logical Error	Medium	Pending
MKTU-3	nextSavedFundingFactorPer Second Cycling	Logical Error	Medium	Pending
CON-1	Lack Of Parameter Validation	Validation	Medium	Pending
GSU-1	tx.gasprice != 0 Is Fallible	Validation	Medium	Pending
OCL-1	Current Ref Price Compared With Earlier Price	Validation	Medium	Pending
GLOBAL-1	Execution Gas Validated Too Early	Validation	Medium	Pending
MKTU-4	Users Paid Funding Fees When They Should Pay	Incentives	Medium	Pending
MKTU-5	Funding Factor Spikes To Max	Logical Error	Medium	Pending
MKTU-6	Unequal Funding fees Over Equal Durations	Logical Error	Medium	Pending
MKTU-7	Funding Fees Used To Brick Market	Logical Error	Low	Pending

Findings & Resolutions

ID	Title	Category	Severity	Status
POSU-1	Not All LiquidatablePosition Args Set	Events	Low	Pending
MKTU-8	Transition From Default To Dynamic Funding	Unexpected Behaviour	Low	Pending
MKTU-9	fundingDecreaseFactorPerSecond Not Checked	Validation	Low	Pending
MKTU-10	Useful Event Data	Events	Low	Pending
BOH-1	Outdated NatSpec	Documentation	Low	Pending
GLOBAL-2	Orders/Deposits/Withdrawals Not Passed As Params	Optimization	Low	Pending
GLOBAL-3	Impact Pool Distribution Dampers Incentives	Incentives	Low	Pending
GSU-2	OOG Check Fallible	Validation	Low	Pending
KEY-1	Naming Conventions Ignored	Naming	Low	Pending

CALC-1 | boundMagnitude Fails To Bound Magnitude

Category	Severity	Location	Status
Logical Error	● Critical	Calc.sol: 27	Pending

Description PoC

When computing the sign of the resulting bounded value, the original value is divided by the magnitude.

However the magnitude has already been adjusted to be within the min or max bounds, therefore when the magnitude has been adjusted the resulting sign variable value is no longer a unit vector indicating sign.

This results in the bounded value, in this case the `nextSavedFundingFactorPerSecond`, being in fact unbounded.

For example:

- value = 800
- Min = 250
- Max = 400
- Magnitude is capped to 400
- Sign = 800 / 400 = 2

The returned value is 400 * 800 / 400 = 800 which is outside of the defined max
As a result, dynamic funding fees cannot be capped, leading to any market being entirely bricked within hours of the dynamic funding fees being activated.

Recommendation

If the value is < 0 return `-magnitude.toInt256()` and if the value is >=0 return `magnitude.toInt256()`

Resolution

MKTU-1 | Funding Factor Per Second Open To Manipulation

Category	Severity	Location	Status
Protocol Manipulation	High	MarketUtils.sol	Pending

Description Video

The adaptive funding rate mechanism is prone to strategic gaming as the resulting `fundingFactorPerSecond` that a market experiences is affected by the amount of time that passes before the funding fees are updated. There are several ways the funding fees may be gamed as a result.

Firstly, consider a trader, Bob, who holds a short position in the following scenario:

- Longs currently pay shorts, long OI > short OI, savedFundingFactorPerSecond > 0.
- Long OI becomes < short OI and the OI diff is within the `NoChange` range.
- The `fundingFactorPerSecond` will begin decreasing, as the skew has changed sides.
- As soon as the `fundingFactorPerSecond` crosses the threshold from positive to negative and becomes `-1`, Bob updates his order by removing 1 wei of collateral.
- Now the `fundingFactorPerSecond` sticks to that minimum magnitude rate, dependent on when Bob updated his position.

The terminal `fundingFactorPerSecond` that the market ends up experiencing is dependent on when the funding fees are updated, therefore Bob updates the funding fees at a time that is most advantageous to him and results in minimal funding paid.

Secondly, consider the following occurs from where we left off with Bob:

- Alice is a long trader, she notices that the funding rate was stopped at the minimum magnitude, so she attempts to increase it.
- Alice opens another temporary long position just large enough to flip the skew back towards longs > shorts, as soon as the `fundingFactorPerSecond` crosses the threshold and becomes `1`, Alice closes her temporary long position.
- Now the `fundingFactorPerSecond` begins decreasing again. Alice is hoping that Bob, or anyone else, will not update the funding — ideally until the `fundingFactorPerSecond` reaches the maximum magnitude.

In a perfectly competitive GMX V2 market, traders on both the long and short side would out-maneuver each other such that the minimum amount of funding fees is paid to the other side at all times. In reality, some traders will abuse these behaviors against uninformed traders to gain an unfair advantage and benefit from substantially less funding fees paid and substantially more funding fees collected.

Recommendation

Consider assigning a `fundingFactoPerSecond` that is halfway between the `min` and `max` magnitude upon switching skews and allowing that middle value to remain, increase, or decrease based on the OI diff.

Additionally, Instead of allowing minute changes to the open interest (e.g. 1 wei) to impact the rate, consider setting a threshold for OI changes to adjust the funding rate. Furthermore, carefully monitor the market for funding fee manipulation and adjust funding factors appropriately.

CALC-2 | boundMagnitude Function Cannot Bound 0

Category	Severity	Location	Status
Logical Error	Medium	Calc.sol: 27	Pending

Description

In the `boundMagnitude` function, when the provided value is 0 the provided min fails to give a lower bound to the resulting value.

- value = 0, min = 10
- sign = value / magnitude = 0 / 10
- Returned value = magnitude * sign = 10 * 0 = 0

Recommendation

When the value is 0 return the min with a sign informed by a boolean parameter. When the `FundingRateChangeType` is Increase, the sign of the resulting min value should be whichever direction the increase is heading in.

Resolution

MKTU-2 | minFundingFactorPerSecond != 0 Undesired behaviors

Category	Severity	Location	Status
Logical Error	Medium	MarketUtils.Sol: 1331	Pending

Description

The funding factor per second returned from the `getNextFundingFactorPerSecond` function may be unable to cross from negative to positive or from positive to negative in the event that the `minFundingFactorPerSecond` is set > 0 and orders are executed often.

- Original `savedFundingFactorPerSecond` = 10
- `minFundingFactorPerSecond` = 10
- `nextSavedFundingFactorPerSecond` = 7
- Bounded `nextSavedFundingFactorPerSecond` = 10

The `nextSavedFundingFactorPerSecond` cannot flip signs and continue to make progress to reach the funding factor it ought to be on the other side unless it can make a large enough jump to cross the gap from [0, `minFundingFactorPerSecond`], which may be unlikely if orders are consistently updating the `secondsSinceFundingUpdated`.

Additionally, if the gap from [0, `minFundingFactorPerSecond`] is crossed successfully, the resulting `nextSavedFundingFactorPerSecond` will be increased in magnitude to the `minFundingFactorPerSecond`, resulting as `-minFundingFactorPerSecond`. This jump will go above the configured `fundingIncreaseFactorPerSecond` rate and may cause unexpected results.

Recommendation

Be wary when configuring the `minFundingFactorPerSecond` to be != 0, if the `minFundingFactor` is ever != 0 it should have a minimal magnitude to limit these behaviors.

Resolution

MKTU-3 | nextSavedFundingFactorPerSecond Cycling

Category	Severity	Location	Status
Logical Error	Medium	MarketUtils.sol: 1322	Pending

Description

The `nextSavedFundingFactorPerSecond` can be decreased to 0 in the event that the `cache.savedFundingFactorPerSecondMagnitude == decreaseValue`.
This can result in the funding side increasing rather than decreasing since it isn't set to 1 or -1:

- Long OI < short OI
- Original `nextSavedFundingFactorPerSecond` = -10
- Original `longsPayShorts` = false
- `decreaseValue` = 10
- `nextSavedFundingFactorPerSecond` = 0
- Long OI < Short OI
- `nextSavedFundingFactorPerSecond` = 0
- `isSkewTheSameDirectionAsFunding` = false
- `increaseValue` = 13
- `nextSavedFundingFactorPerSecond` = -13

Therefore the `fundingFactorPerSecond` was meant to decrease in magnitude but increased instead.

Recommendation

Avoid this cycling by assigning the `nextSavedFundingFactorPerSecond` to 1 or -1 in the `if` case on line 1316 by changing the `cache.savedFundingFactorPerSecondMagnitude < decreaseValue` to `cache.savedFundingFactorPerSecondMagnitude <= decreaseValue`.

Resolution

CON-1 | Lack Of Parameter Validation

Category	Severity	Location	Status
Validation	Medium	Config.sol	Pending

Description

There is no validation that the `THRESHOLD_FOR_DECREASE_FUNDING` is less than `THRESHOLD_FOR_STABLE_FUNDING`. If the config keeper were to invert the thresholds, the funding factor may be decreasing when it should be stable or increasing when it should be decreasing.

Furthermore, there is no validation that the `MIN_FUNDING_FACTOR_PER_SECOND` is less than the `MAX_FUNDING_FACTOR_PER_SECOND`. As a result, `boundMagnitude` will bound the value incorrectly.

Recommendation

Add validation in the Config to ensure `THRESHOLD_FOR_DECREASE_FUNDING` is less than `THRESHOLD_FOR_STABLE_FUNDING`.

Add validation in the Config to ensure `MIN_FUNDING_FACTOR_PER_SECOND` is less than `MAX_FUNDING_FACTOR_PER_SECOND`. A check for the min and max bounds may be included in `boundMagnitude` as well or the documentation should mention that the validation is done elsewhere.

Resolution

GSU-1 | tx.gasprice != 0 Is Fallible

Category	Severity	Location	Status
Validation	Medium	GasUtils.sol: 80	Pending

Description

`tx.gasprice != 0` is not a bulletproof means of filtering out non-estimateGas calls. The Keeper can assign a non-zero `gasPrice` and there has been discussion about getting the `tx.gasprice` to be the basefee even when the `gasPrice = 0` in an `estimateGas` call.

Recommendation

Use `tx.origin` as no one can transact from the zero address.

Resolution

OCL-1 | Current Ref Price Compared With Earlier Price

Category	Severity	Location	Status
Validation	Medium	Oracle.sol: 744-759	Pending

Description

A keeper may have to use prices from several blocks ago to execute an order, regardless of if realtime feeds or the default oracle system is being used. In such a case, the `validateRefPrice` would be comparing the latest Chainlink aggregator oracle price against an earlier price.

Depending on how large the `MAX_ORACLE_REF_PRICE_DEVIATION_FACTOR` is and how volatile the asset, the execution with these earlier block numbers/prices may revert, preventing them from being used.

Recommendation

Carefully assign the `MAX_ORACLE_REF_PRICE_DEVIATION_FACTOR` considering that it might be necessary in some cases to allow prices from blocks previous to when the `latestAnswer` in the Chainlink aggregator was updated.

Resolution

GLOBAL-1 | Execution Gas Validated Too Early

Category	Severity	Location	Status
Validation	Medium	Global	Pending

Description

Based on the `startingGas` and the `estimatedGasLimit`, the execution gas is validated to ensure the `startingGas` is greater than the `estimatedGasLimit` and some variable, additional gas for execution. After calling `GasUtils.validateExecutionGas(dataStore, startingGas, estimatedGasLimit)`, the gas for execution is presumed to be enough for execution of the order as it has been validated.

However, `uint256 executionGas = GasUtils.getExecutionGas(dataStore, startingGas)` is called right afterwards which reduces the `startingGas` by the gas needed for error handling. The amount of gas validated for execution is different than the amount given for execution, which may now be insufficient.

Recommendation

Consider making the gas validation more restrictive by calling `validateExecutionGas()` on the result of `getExecutionGas()` or ensure the the `minAdditionalGasForExecution` is large enough to cover the gas forwarded to handle the execution error.

Resolution

MKTU-4 | Users Paid Funding Fees When They Should Pay

Category	Severity	Location	Status
Incentives	Medium	MarketUtils.sol: 1235	Pending

Description PoC

It is possible for shorts to get paid when longs should pay shorts (long OI > short OI) and vice versa.

This is because it may take a certain duration until the `savedFundingFactorPerSecond` flips payment sides and accurately represents payment direction. Ultimately this functionality misaligns incentives during this period, where traders who imbalance the pool aren't discouraged by funding fee payments.

Recommendation

Consider resetting the `fundingFactorPerSecond` entirely when the OI imbalance direction changes.

Resolution

MKTU-5 | Funding Factor Spikes To Max

Category	Severity	Location	Status
Logical Error	Medium	MarketUtils.sol: 1301	Pending

Description

Because the `increaseValue` is dependent on `durationInSeconds`, when a period of time goes by without updates the `cache.nextSavedFundingFactorPerSecond` can spike to the maximum bound. This can occur when the skew changes, causing an increase to a large `fundingFactoPerSecond` regardless of the new OI diff.

The situation also may arise if the `savedFundingFactorPerSecond` is at 0 and the duration is large, regardless of whether the previous `fundingRateChangeType` was an increase or no change.

Recommendation

It's crucial to track the min/max limits and make adjustments as needed. Additionally, consider refactoring the funding such that these spikes do not occur when the skew is switching sides or the `savedFundingFactorPerSecond` is 0.

Resolution

MKTU-6 | Unequal Funding fees Over Equal Durations

Category	Severity	Location	Status
Logical Error	Medium	MarketUtils.sol	Pending

Description PoC

Traders don't experience an incremental increase in the `fundingFactorPerSecond`. Instead, they receive funding based on the final funding factor applicable for the entire duration of their position. Consequently, a user who updates their position after X seconds will receive funding fees based on the rate at that specific moment.

This could lead to a scenario where updating a position at the end of X seconds may yield more in funding fees than updating midway at X/2 and closing out another X/2 later, even though the total duration is the same.

Recommendation

Clearly document this behavior so that traders are aware how frequent updates can affect funding fee payments.

Resolution

MKTU-7 | Funding Fees Used To Brick Market

Category	Severity	Location	Status
Logical Error	● Low	MarketUtils.sol: 2804	Pending

Description PoC

The `claimableFundingFeeAmount` is asserted to be less than the balance of the `MarketToken` contract at all times, however this may not always be the case and can lead to the halting of liquidations, orders, ADLs, and withdrawals as a result.

Firstly, users may decrease the collateral backing their position to 0 given that the `willPositionCollateralBeSufficient` validation ignores fees and price impact.
All funding fees accumulated by positions with 0 collateral will not have corresponding ERC20 tokens backing that funding fee amount. Therefore the funding fees will grow without a corresponding backing value in the balance of the `MarketToken`.

A malicious actor may then swap all (or most) of the backing `poolAmount` to the other token in the market. This way the attacker can reduce the amount of ERC20 tokens that can count towards the `balance` in the `validateMarketTokenBalance` validation to just over the `claimableFundingFeeAmount`. The funding fees will then continue to increase and surpass the balance of the contract. Thereby causing orders for positions that would collect these funding fees as a `claimableFundingFeeAmount` to revert.

This exploit is unlikely as it requires a market with low open interest and would require time to execute as well as favorable funding rates. Additionally it could be resolved by manually plugging the hole by sending an amount of ERC20 tokens directly to the `MarketToken`. However it could be leveraged by an attacker for a short period of time to cause grief to other users in the same market before tokens are swapped back.

Recommendation

Consider removing the `claimableFundingFeeAmount` validation in the `validateMarketTokenBalance` function entirely. Otherwise carefully monitor for any such manipulation.

Resolution

POSU-1 | Not All LiquidatablePosition Args Set

Category	Severity	Location	Status
Events	● Low	PositionUtils.sol: 328	Pending

Description

If a position is deemed liquidatable before validating against `info.minCollateralUsdForLeverage`, then any `LiquidatablePosition` events will emit 0 as the `minCollateralUsdForLeverage` since `info.minCollateralUsdForLeverage` has yet to be set.

Recommendation

Document such behavior or calculate the `minCollateralUsdForLeverage` if needed.

Resolution

MKTU-8 | Transition From Default To Dynamic Funding

Category	Severity	Location	Status
Unexpected Behaviour	● Low	MarketUtils.sol: 1268	Pending

Description

In the `getNextFundingFactorPerSecond` function, when the `fundingIncreaseFactorPerSecond` is not configured the `savedFundingFactorPerSecond` is assigned to 0.

However for markets where the default funding is used before activating the dynamic funding it may serve as a smoother transition to assign the `savedFundingFactorPerSecond` to the current resulting `fundingFactorPerSecond`.

Recommendation

Provide the resulting `fundingFactorPerSecond` as the returned `nextSavedFundingFactorPerSecond` value on line 1268.

Resolution

MKTU-9 | fundingDecreaseFactorPerSecond Not Checked

Category	Severity	Location	Status
Validation	Low	MarketUtils.sol: 1261	Pending

Description

In the `getNextFundingFactorPerSecond` function, only `configCache.fundingIncreaseFactorPerSecond` is validated against in `if (configCache.fundingIncreaseFactorPerSecond == 0)`.

It is possible for the `fundingIncreaseFactorPerSecond` to be non-zero, but the `fundingDecreaseFactorPerSecond` to be zero. As a result, the `decreaseValue` would be be zero and the saved funding factor would not be adjusted -- mimicking a stable behavior when it should really decrease.

This could return a different result than simply calculating the funding factor per second using `Precision.applyFactor(cache.diffUsdToOpenInterestFactor, cache.fundingFactor)`.

Recommendation

Validate the funding factor depending on the adjustment direction and/or document this behavior.

Resolution

MKTU-10 | Useful Event Data

Category	Severity	Location	Status
Events	● Low	MarketUtils.sol: 2384	Pending

Description

In the `distributePositionImpactPool` function the `distributionAmount` is emitted with the `“PositionImpactPoolDistributed”` event.

However it may be helpful for consumers of the `“PositionImpactPoolDistributed”` event to have access to the resulting value of the position impact pool as well.

Recommendation

Consider adding the `nextPositionImpactPoolAmount` to the information emitted with the `“PositionImpactPoolDistributed”` event.

Resolution

BOH-1 | Outdated NatSpec

Category	Severity	Location	Status
Documentation	● Low	BaseOrderHandler.sol: 63	Pending

Description

The current NatSpec does not include the new `Order.Props memory order` parameter.

Recommendation

Update the documentation.

Resolution

GLOBAL-2 | Orders/Deposits/Withdrawals Not Passed As Params

Category	Severity	Location	Status
Optimization	● Low	Global	Pending

Description

During cancellation, the Order/Deposit/Withdrawal is read from the store just to validate the request cancellation, and then the appropriate cancel function is called in a Utils library. In Utils, the Order/Deposit/Withdrawal is read once again from the store.

Recommendation

Consider passing the objects as parameter which matches the updates in the rest of the codebase.

Resolution

GLOBAL-3 | Impact Pool Distribution Dampers Incentives

Category	Severity	Location	Status
Incentives	● Low	Global	Pending

Description

Positive position impact is a key incentive for actors to rebalance the long and short open interest of a market. However positive position impact is capped by the amount of virtual index tokens in the position impact pool.

When the position impact pool is distributed, the maximum net cap for positive impact is reduced. If the `minPositionImpactPoolAmount` is not configured high enough it could lead to lacking incentives for users to rebalance the open interest.

Recommendation

Ensure that the `minPositionImpactPoolAmount` is configured to a reasonable amount, allowing for sufficient incentivization for aeros to balance the open interest.

Resolution

GSU-2 | OOG Check Fallible

Category	Severity	Location	Status
Validation	● Low	GasUtils.sol: 80	Pending

Description

In the `validateExecutionErrorGas` function, the revert `reasonBytes` is required to be of length 0 to initiate the validation. However the length of the `reasonBytes` may be 0 in many circumstances other than an out of gas error. For example if an empty `revert()` or `require()` is used without a revert string, or another type of error such as an `INVALID` opcode.

Recommendation

Be aware that the `reasonBytes.length == 0` check does not ensure that the execution reverted with an out of gas error and monitor for any potential manipulations because of this assumption.

Resolution

KEY-1 | Naming Conventions Ignored

Category	Severity	Location	Status
Naming	● Low	Keys.sol: 1288	Pending

Description

The `minMarketTokensForFirstDeposit` function does not follow the established naming convention for functions used to retrieve keys. The expected name for this function would be `minMarketTokensForFirstDepositKey`.

Recommendation

Rename the `minMarketTokensForFirstDeposit` function to `minMarketTokensForFirstDepositKey`.

Resolution

Disclaimer

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

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

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

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

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

About Guardian Audits

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

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

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

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