

ETHEREAL · SECURITY REVIEW

Ethereal Protocol Review.

A security review of Protocol Review.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Ethereal	Protocol Review	Jakub Zmyslowski, Luke Selvan	July 20 – August 6, 2026	August 6, 2026

§ 00

Contents.

59 pages: overview, confidence ranking, scope, 42 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 07
	Main Review · 42 findings	P. 09
§ 05	Disclaimer	P. 58
§ 06	About Guardian	P. 59

§ 01

Overview.

Guardian reviewed Protocol Review across 2 review rounds. The final report includes 1 High, 4 Medium, 15 Low, 22 Informational findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/GuardianOrg/merkl-contracts-team1-1784749189571 , https://github.com/meridianxyz/contracts
CLIENT	METHODOLOGY
Ethereal	Smart contract · Manual · Main Review · Remediation Review
PROJECT	MAIN REVIEW COMMIT
Protocol Review	b7853b931ec0
PREPARED BY	
Jakub Zmyslowski, Luke Selvan	
AUDIT WINDOW	
July 20 – August 6, 2026	
FINAL REPORT	
August 6, 2026	

1.1 Confidence Ranking

4 / 5

High Confidence

Guardian assigns a Confidence Ranking of 4 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
■ Critical	0	0	0	0	0
■ High	1	0	0	0	1
■ Medium	4	0	2	0	2
■ Low	15	3	8	0	4
■ Informational	22	8	7	0	7
All severities	42	11	17	0	14

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence ASSIGNED	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed Protocol Review in

<https://github.com/GuardianOrg/merkl-contracts-team1-1784749189571>,
<https://github.com/meridianxyz/contracts>. The scope includes 106 source lines across 1 file.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

Files in Scope.

FILE	NSLOC	TOTAL
contracts/partners/tokenWrappers/MeridianDepositWrapper.sol	106	180

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
H-01	Short EIP1271 Return Halts processActions Queue	DoS	HIGH	RESOLVED	p. 9
M-01	Pyth Freshness Check Uses Wrong Units	Oracle	MEDIUM	RESOLVED	p. 10
M-02	Fresh Payloads Can Contain Stale Prices	Oracle	MEDIUM	ACKNOWLEDGED	p. 11
M-03	Manipulable Delist Settlement Price	Oracle	MEDIUM	ACKNOWLEDGED	p. 12
M-04	Delisting Can Block the Relay Queue	Logical Error	MEDIUM	RESOLVED	p. 13
L-01	Default Switch Exposes New Backing	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 15
L-02	Maximum ID Disables Claimer Batches	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 16
L-03	Claimer Traps OFT Fee Refunds	Logical Error	LOW	ACKNOWLEDGED	p. 17
L-04	Inconsistent zero-price guard checks	Validation	LOW	RESOLVED	p. 18
L-05	Realized PnL Mismatch Events Are Never Emitted	Events	LOW	RESOLVED	p. 19
L-06	Zero Fee Recipient Blocks New Campaigns	DoS	LOW	RESOLVED	p. 20
L-07	Zero Leverage Creates Unusable Products	Validation	LOW	RESOLVED	p. 21
L-08	Expiry Updates Extend Existing Signatures	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 22
L-09	Updates During Delisting Mint Unbacked Balance	Best Practices	LOW	ACKNOWLEDGED	p. 23
L-10	Partial Deleverage Blocks All Liquidations	DoS	LOW	ACKNOWLEDGED	p. 25
L-11	Timestamp Skew Causes Wrongful Liquidation	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 28
L-12	Stale ADL Tolerance Can Lock Liquidations	Logical Error	LOW	PENDING	p. 30
L-13	Deferred PnL Drift Can Freeze ADL	Unexpected Behavior	LOW	PENDING	p. 31
L-14	Token Removal Ignores Pending Withdrawals	Logical Error	LOW	ACKNOWLEDGED	p. 33
L-15	Missed intra-user equity peaks can indefinitely disable all liquidations while the backstop is solvent	Informational	LOW	PENDING	p. 34

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-01	Liquidations Always Seize A User's Entire Margin	Suggestion	INFORMATIONAL	ACKNOWLEDGED	p. 35
I-02	Signed Oracle Prices Decode as Unsigned	Unexpected Behavior	INFORMATIONAL	RESOLVED	p. 36
I-03	Missing View Modifiers in Gateway Interface	Informational	INFORMATIONAL	RESOLVED	p. 37
I-04	Duplicate Using Directives In PerpEngine	Informational	INFORMATIONAL	RESOLVED	p. 38
I-05	Unsupported Exponents Halt Risk Actions	Suggestion	INFORMATIONAL	ACKNOWLEDGED	p. 39
I-06	Packed ADL Fields Limit Settlements	Documentation	INFORMATIONAL	RESOLVED	p. 40
I-07	Position Fee Rebates Are Not Supported	Suggestion	INFORMATIONAL	PENDING	p. 41
I-08	Zero Max Quantity Renders A Product Close-Only	Validation	INFORMATIONAL	RESOLVED	p. 42
I-09	Disabling The Mark Price Cap Is Dangerous	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 43
I-10	Emergency Pause Blocks Liquidations	Warning	INFORMATIONAL	ACKNOWLEDGED	p. 44
I-11	Zero-valued withdrawal request ID conflicts with the fresh Claimer cursor	Informational	INFORMATIONAL	PENDING	p. 45
I-12	Token configuration NatSpec incorrectly labels D9 values as native units	Informational	INFORMATIONAL	RESOLVED	p. 46
I-13	Zero-normalized deposits register collateral-free subaccounts	Informational	INFORMATIONAL	PENDING	p. 47
I-14	Expired partial-order fills are never pruned	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 48
I-15	Revoked linked signer addresses cannot later become trader accounts	Informational	INFORMATIONAL	PENDING	p. 49
I-16	Synthetic token capacity is accepted but has no defined effect	Informational	INFORMATIONAL	PENDING	p. 50
I-17	Per-account caps cannot prevent narrow ADL recovery from freezing all liquidations	Informational	INFORMATIONAL	PENDING	p. 51
I-18	Local withdrawals silently discard low-decimal collateral remainders created by normal trading	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 53
I-19	Deposit Capacity Behavior Is Undocumented	Documentation	INFORMATIONAL	RESOLVED	p. 54
I-20	Pending withdrawals bypass the token-specific completion halt	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 55
I-21	Binary64 tolerance can admit liquidation marginally above exact maintenance	Informational	INFORMATIONAL	PENDING	p. 56
I-22	Tolerated IEEE-754 notional drift pollutes position basis	Informational	INFORMATIONAL	PENDING	p. 57

HIGH **H-01**

RESOLVED

Short EIP1271 Return Halts processActions Queue

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/lib/VerifierLib.sol#L126
REVIEW	Main Review

DESCRIPTION

The `isValidSignature` function of the `VerifierLib` contract must never revert, since the `processActions` loop relies on it returning a boolean when handling `_handleLinkSigner` and `_handleRevokeLinkedSigner` actions. Both handlers invoke it with `shouldRevert` set to false and treat a false result as a clean rejection rather than a failure.

Despite this, `VerifierLib._verifyEip1271Signature` can be pushed into a revert by a contract that exposes an `isValidSignature` function crafted to answer with a short payload. Once the staticcall returns, the check `success := and(success, gt(returndatasize(), 0))` filters out only empty responses, so a reply of 1 to 3 bytes survives and proceeds to the copy step.

At that point the code runs `returndatacopy(add(returnData, 0x20), 0, bytesToCopy)` with `bytesToCopy` fixed at 4. Whenever `returndatasize` is 1, 2, or 3, the read satisfies `offset + size > returndatasize` and goes out of bounds, causing an exceptional halt.

Since this halt fires within the `VerifierLib` frame itself and not inside the wallet's staticcall, the `shouldRevert` logic never sees it and the error bubbles up unhandled. That breaks the no-revert guarantee of `isValidSignature` and forces the whole `ActionHandler.processActions` batch to revert.

RECOMMENDATION

Only proceed to copy when the response carries the full 4 magic bytes, so any shorter reply is classified as an invalid signature and continues through the existing graceful path. Replace `success := and(success, gt(returndatasize(), 0))` with `success := and(success, gt(returndatasize(), 3))`.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/348>

MEDIUM **M-01**

RESOLVED

Pyth Freshness Check Uses Wrong Units

CATEGORY Oracle

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/758b8a8bc4ee7c54dc97809eac8a3d5d6f514836/src/lib/LiquidationSharedLib.sol#L80-L82>

REVIEW Main Review

DESCRIPTION

`LiquidationSharedLib.parsePythPriceFeed` compares Pyth's signed update timestamp directly with `block.timestamp`.

```
``plain text
if (block.timestamp - maxAge > data.timestamp) {
  revert PriceTooOld(id, data.timestamp, uint64(block.timestamp));
}
```

However, Pyth encodes this timestamp in microseconds, while `block.timestamp` and `pythMaxAge` are denominated in seconds. Consequently, the comparison in `LiquidationSharedLib` cannot reject realistically encoded stale payloads. `pythMaxAge` therefore imposes no effective age limit while the payload's signer remains authorized.

A stale price can incorrectly determine whether an account is below maintenance margin or whether the liquidator is bankrupt.

RECOMMENDATION

Normalize both values to the same unit before comparing them. Prefer converting `block.timestamp` and `pythMaxAge` to microseconds to avoid discarding timestamp precision.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/347>

MEDIUM **M-02**

ACKNOWLEDGED

Fresh Payloads Can Contain Stale Prices

CATEGORY	Oracle
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/758b8a8bc4ee7c54dc97809eac8a3d5d6f514836/src/lib/external/pyth/PythLazerLib.sol#L7-L14 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/758b8a8bc4ee7c54dc97809eac8a3d5d6f514836/src/lib/LiquidationSharedLib.sol#L35-L84
REVIEW	Main Review

DESCRIPTION

Pyth Lazer's payload header contains `timestampUs`, while each feed exposes a separate `feedUpdateTimestamp` identifying when its price was generated. Pyth's [Payload Reference] (<https://docs.pyth.network/price-feeds/pro/payload-reference>) explains that prices may be carried forward during market closures. Its [price-data documentation](<https://docs.pyth.network/price-feeds/pro/understanding-price-data#price>) also explains that prices are carried forward when insufficient publishers contribute. Consumers must therefore validate `feedUpdateTimestamp` rather than relying solely on the header timestamp.

Meridian's parser does not support `feedUpdateTimestamp`. Consequently, even after correcting the header timestamp's unit, a recently signed payload can contain a price generated hours or days earlier and still pass validation.

For example, when a market reopens after a weekend, a fresh XAU payload may continue carrying Friday's price until Pyth generates a new aggregate. If the market has moved favorably for a position, the carried price can make an otherwise healthy account appear below maintenance margin and permit wrongful liquidation. It can likewise make a solvent liquidator appear bankrupt and incorrectly authorize ADL.

RECOMMENDATION

Update PythLazerLib to support `FeedUpdateTimestamp` and modify `parsePythPriceFeed` to require and validate it. Ensure submitted liquidation and ADL payloads include the property. Reject a payload when the property is absent, later than the header timestamp, or older than `pythMaxAge` after normalizing all timestamps to the same unit.

RESOLUTION

Status: Acknowledged.

<https://github.com/meridianxyz/contracts/commit/c2927c7e204573a04976434bcf1094a1926d959b>, <https://github.com/meridianxyz/contracts/commit/393f87ec19ef467f167711a0eb9d99a97d722dab>

MEDIUM **M-03**

ACKNOWLEDGED

Manipulable Delist Settlement Price

CATEGORY	Oracle
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L157-L164
REVIEW	Main Review

— DESCRIPTION

When a perp is delisted, `PerpEngine.initiateDelist` reads a single Pyth Lazer price via `LiquidationSharedLib.parsePythPriceFeed` and stores it as `product.delistExpiryPrice`, the price at which every position is force-closed in `PerpEngine.settleDelistPosition`. Since settlement uses one instantaneous snapshot instead of a time-weighted average, that single price can be manipulated to lock in PnL on the attacker's own position, forcing every counterparty to be settled at that price.

Attack path:

1. The attacker learns the time an asset will be delisted from a public announcement
 2. They build a large long position on the perp.
 3. Just before the delist time, they buy heavily on spot exchanges to sharply inflate the spot price for a short period.
 4. The delist is called and the inflated oracle price is stored as `delistExpiryPrice`.
 5. They sell back the spot they bought, while their perp profit stays locked in at the inflated settlement price.
- Note that this attack is only realistic for lower-liquidity tokens where the spot price can be moved significantly cheaply, but those are also the assets most likely to be delisted in the first place.

— RECOMMENDATION

Settle delists on a time-weighted average of the oracle price over a fixed window, as [Hyperliquid] (<https://hyperliquid.gitbook.io/hyperliquid-docs/trading/delisting>) and other exchanges do to mitigate this issue.

— RESOLUTION

Status: Acknowledged.

MEDIUM **M-04**

RESOLVED

Delisting Can Block the Relay Queue

CATEGORY Logical Error

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L295-L307> <https://github.com/GuardianOrg/sequencer-team1-1782406004832-team1-1784555817417/blob/6944f4c4be70f4bb6e48c44021312cfbccf4930d/src/modules/root/relayer/relayer.service.ts#L360-L384>

REVIEW Main Review

DESCRIPTION

Product delisting, funding updates, and position-fee updates are submitted as separate sequencer actions.

`PerpEngine.finalizeDelist` permanently changes the product status to `DELISTED`, while both `PerpEngine.updateFunding` and `PerpEngine.updatePositionFee` only accept active products.

```
``plain text
```

```
if (product.status !== ProductStatus.ACTIVE) {
  revert ProductStatusViolation(id, productId, EngineType.PERP, product.status);
}
```

A funding request submitted before engine expiry can be confirmed after the expiry message has already generated `FinalizePerpProductDelist`. `FundingHandler` still applies this late confirmation and enqueues an `UpdateFunding` action without checking that the product has been delisted. The funding action therefore receives a higher action ID than the finalization action and can no longer execute once finalization is processed.

The same applies to position fees. After engine expiry but before the sequencer processes the expiry message, the position-fee worker can submit a charge while the database still considers the product active. `Nekuti` accepts the charge, and `PositionFeeHandler` subsequently enqueues `UpdatePositionFee` without checking the product status. The position-fee action therefore also receives a higher action ID than the finalization action.

Consequently, if the sequencer submits `FinalizePerpProductDelist` before an already-generated `UpdateFunding` or `UpdatePositionFee` action, the update can no longer execute. If both actions are included in one transaction, the entire batch reverts atomically and delist finalization is rolled back. If finalization is submitted separately first, the subsequent update reverts without changing state.

`ActionHandler` requires action IDs to be strictly increasing but not contiguous, meaning the sequencer can omit the obsolete action and continue with a higher action ID. However, the current off-chain relayer always selects the oldest pending action and provides no rejected or discarded state. Once the obsolete update reaches the head of the queue, the relayer worker terminates on every attempt and selects the same action again after restarting. Consequently, all later actions (including order settlement, deposit finalization,

M-04 Delisting Can Block the Relay Queue

CONTINUED

withdrawal initiation, liquidation, and ADL) remain pending, causing off-chain/on-chain divergence and allowing unhealthy positions to accumulate further losses.

— RECOMMENDATION

Mark the product as `DELISTING` before requesting engine expiry, block new funding and position-fee charges, and reconcile all in-flight charges.

— RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/sequencer/pull/1696>

LOW **L-01**

ACKNOWLEDGED

Default Switch Exposes New Backing

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/external/MeridianUSD.sol#L60-L94 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/README.md#L323-L338
REVIEW	Main Review

DESCRIPTION

MeridianUSD supports multiple approved backing tokens, but `wrap` and `unwrap` accept only `defaultWrappedToken`. Meanwhile, `setDefaultWrappedToken` verifies only that the new token is enabled - it neither migrates reserves nor requires the contract's balance of the new default to cover the outstanding merUSD supply.

The [documented migration procedure](<https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/README.md#L323-L338>) requires wrapping the new asset and unwrapping the old asset before changing the default. However, this cannot execute because `wrap(newToken)` reverts while the old token remains the default.

Consequently, the owner can legitimately switch from A to an empty B reserve while A still backs all existing merUSD. Subsequent B deposits cannot repair this deficit because each deposit increases the B reserve and merUSD supply by the same amount.

Any existing merUSD holder can redeem the newly deposited B before the new depositor. The depositor remains with merUSD backed by A, but cannot redeem A while B is the default.

If A is depegged or frozen, legacy holders can escape its loss using new depositors' B. Even when A remains sound, withdrawals fail after B is exhausted until the owner intervenes.

RECOMMENDATION

Consider adding an atomic owner-only migration function that pulls sufficient new backing, verifies actual receipt, switches the default, and then recovers the old reserve without changing merUSD supply.

RESOLUTION

Status: Acknowledged.

LOW **L-02**

ACKNOWLEDGED

Maximum ID Disables Claimer Batches

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0e4eb953574540e4a10b8e0ad4f1e/src/Claimer.sol#L92-L120
REVIEW	Main Review

DESCRIPTION

`Claimer.finalizeWithdrawals` accepts caller-supplied uint64 request IDs and requires only that each ID is greater than the previously processed value. The ID is advanced regardless of whether the underlying withdrawal succeeds.

An authorized withdrawal claimer can accidentally, or if compromised, submit `type(uint64).max` with an invalid account or withdrawal digest. The underlying finalization fails, but the outer batch succeeds and stores the maximum ID.

No future request can provide a larger uint64, so every subsequent `finalizeWithdrawals` call on that Claimer instance reverts.

Individual withdrawals remain directly finalizable through ExchangeGateway, but automated batching requires upgrading or replacing the Claimer contract.

RECOMMENDATION

Consider requiring request IDs to be consecutive rather than accepting any larger value, deriving the expected ID from the stored cursor.

RESOLUTION

Status: Acknowledged.

LOW **L-03**

ACKNOWLEDGED

Claimer Traps OFT Fee Refunds

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/Claimer.sol#L97-L125 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/CollateralManager.sol#L296-L366
REVIEW	Main Review

DESCRIPTION

`Claimer.finalizeWithdrawals` forwards each caller-funded `oftSendFee` budget to `ExchangeGateway`. For a successful cross-chain withdrawal, `CollateralManager._completeWithdrawal` obtains the current LayerZero fee, spends only that amount, and refunds the unused budget to `msg.sender`. Because the call originated from Claimer, the refund is returned to the Claimer contract.

However, `finalizeWithdrawals` only refunds the `oftSendFee` allocated to unsuccessful finalization attempts and any value supplied above the total declared budgets.

```
``plain text
if (amountToRefund > 0) {
  SafeTransferLib.safeTransferETH(msg.sender, amountToRefund);
}
```

ETH returned by successful calls is not included. If a request declares budget `B` but the current fee is `F`, the operator pays `B` while `B - F` remains in Claimer. This can occur when the LayerZero fee decreases between the off-chain quote and transaction execution.

This balance cannot fund future batches because each call requires sufficient new `msg.value`. Since Claimer has no ETH recovery function, accumulated overpayments remain inaccessible in the contract.

RECOMMENDATION

Consider snapshotting Claimer's balance excluding the current `msg.value`, then refund the complete balance increase after processing the batch, or adding an owner-only recovery function for ETH already trapped in the contract.

RESOLUTION

Status: Acknowledged.

LOW **L-04**

RESOLVED

Inconsistent zero-price guard checks

CATEGORY	Validation
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L159-L161
REVIEW	Main Review

DESCRIPTION

The `parsePythPriceFeed` function from the `LiquidationSharedLib` contract can return a price of zero. The `data.price` field is zero-initialized and assigned only inside the `Price` property branch, so a payload omitting that property returns zero rather than reverting.

The `initiateDelist` function from the `PerpEngine` contract rejects this with `if (expiryPrice == 0) revert PriceZero(id)`. The four other consumers of the parsed price apply no such check:

`liquidateSubaccount` and `_processLiquidateSubaccountWithCompensationProduct` from the `Liquidation` contract, and `_prepareDeleverageProduct` from both the `Deleverage` and `DeleverageWithCompensation` contracts.

RECOMMENDATION

Apply the same `if (markPrice == 0) revert PriceZero(id)` check at each of the four sites, matching the existing guard in `initiateDelist`.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/355>

LOW **L-05**

RESOLVED

Realized PnL Mismatch Events Are Never Emitted

CATEGORY	Events
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/Deleverage.sol#L814-L818
REVIEW	Main Review

DESCRIPTION

The `_verifyRealizedPnl`, `_verifyLiquidatorRealizedPnl`, and user-side settlement checks in the `PerpEngine`, `Liquidation`, `Deleverage`, and `DeleverageWithCompensation` contracts are intended to emit a mismatch event (e.g. `Deleverage.DeleverageUserRealizedPnlMismatch`) when the sequencer's reported realized PnL drifts from the on-chain reference by a notable-but-tolerable amount, feeding off-chain drift monitoring.

Each site reverts when `absDiff > tolerance` and otherwise emits when `absDiff > 1 gwei`, but every `tolerance` passed is at most `WORST_CASE_REALIZED_PNL_ERROR` (`2_220_448 wei` $\approx$ `0.00222 gwei`), so reaching the `else` branch guarantees `absDiff` is already far below `1 gwei` and the emit branch is dead code.

Consequently these mismatch events are never emitted, leaving any off-chain monitoring that relies on them blind to matching-engine PnL drift.

Affected sites: `Deleverage` L814, L842, L916 · `DeleverageWithCompensation` L909, L937, L1007 · `PerpEngine` L589 · `Liquidation` L813

RECOMMENDATION

Replace the `absDiff > 1 gwei` guard at each of the eight sites with a threshold below the tolerance, such as `absDiff > 0`.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/355>

LOW **L-06**

RESOLVED

Zero Fee Recipient Blocks New Campaigns

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/merkl-contracts-team1-1784749189571/blob/b7853b931ec08b1044d78291d13f45b27df506fc/contracts/partners/tokenWrappers/MeridianDepositWrapper.sol#L188
REVIEW	Main Review

DESCRIPTION

The `MeridianDepositWrapper` stores Merkl's fee recipient, reading `DistributionCreator.feeRecipient` in its constructor and in `setFeeRecipient`. Campaigns using the wrapper are funded through the `DistributionCreator`, which charges a fee in the reward token at creation. Merkl can set `DistributionCreator.feeRecipient` to `address(0)`, a supported state where the `DistributionCreator` contract keeps the fee.

The `DistributionCreator` [remaps a zero recipient to](<https://github.com/AngleProtocol/merkl-contracts/blob/d5d4590bd8c7af6d00385081c6fcdf7b8dfdae2f/contracts/DistributionCreator.sol#L630>) `address(this)` before charging the fee, but the wrapper rejects zero: its constructor and `setFeeRecipient` both revert when `feeRecipient` is `address(0)`, so it keeps a stale recipient. Because `_beforeTokenTransfer` only mints when `to` is the `distributor` or the cached `feeRecipient`, the fee transfer to the `DistributionCreator` mints nothing and reverts on the creator's zero wrapper balance. As a result no new Merkl campaigns can be created while `feeRecipient` is zero. Existing campaigns with a non-zero `feeRecipient` are unaffected.

RECOMMENDATION

In the constructor and `setFeeRecipient`, set `feeRecipient` to the `distributionCreator` address when `DistributionCreator.feeRecipient` returns `address(0)`.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/merkl-contracts/commit/b8ff212426ef4e6ff5c65afd65e8633cfe5db3280>

LOW **L-07**

RESOLVED

Zero Leverage Creates Unusable Products

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngineAdmin.sol#L234-L238>

REVIEW Main Review

DESCRIPTION

The owner can accidentally register an active perpetual product with `maxLeverage` equal to zero because `_verifyMaxLeverage` only enforces the upper bound.

```
``plain text
if (maxLeverage > PRODUCT_MAX_LEVERAGE) {
revert InvalidParameter("maxLeverage", P_ERR_AMT_GT_MAX);
}
`[
```

The registration succeeds on-chain, but the off-chain database requires `max_leverage > 0` and rejects the corresponding indexer insertion.

The database rejection also prevents the indexer checkpoint from being saved, so the indexer remains stuck on the registration block.

The product therefore exists on-chain but not off-chain, while subsequent events cannot be indexed. A positive leverage update in a later block does not automatically resolve the issue because the registration handler continues reading the product's zero leverage at the original registration block.

Additionally, the owner can set `maxLeverage` to zero for an active product that already has open positions. Both liquidation variants divide by `maxLeverage`` and consequently revert, preventing those positions from being liquidated until a positive value is restored.

RECOMMENDATION

Require `maxLeverage` to be greater than zero during product registration, activation, and every leverage update.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/355>

LOW **L-08**

ACKNOWLEDGED

Expiry Updates Extend Existing Signatures

CATEGORY Unexpected Behavior

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/lib/storage/Exchange.sol#L69-L76>

REVIEW Main Review

DESCRIPTION

User signed actions commit only to `signedAt`, while their deadline is calculated at execution using the current owner-configured buffer:

```
``plain text
if (block.timestamp > signedAt + self.actionExpiryBuffers[action]) {
revert ActionExpired(id, block.timestamp, signedAt, self.actionExpiryBuffers[action]);
}
```

Because the buffer is not included in the signed message, changing it changes the validity of outstanding signatures. Increasing the buffer can revive signatures that users considered permanently expired. If an expired action is submitted, execution reverts before consuming its nonce or action ID, so the same action can succeed later if the owner increases the buffer.

Consequently, an authorized submitter holding an old withdrawal, conversion, or other signed request may execute it after its original validity period, causing unexpected asset movements based on stale user intent.

RECOMMENDATION

Consider including an explicit deadline or configuration version in every signed message so later buffer updates cannot change existing signatures' validity.

RESOLUTION

Status: Acknowledged.

LOW **L-09**

ACKNOWLEDGED

Updates During Delisting Mint Unbacked Balance

CATEGORY Best Practices

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L311>, <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L295>

REVIEW Main Review

DESCRIPTION

`updatePositionFee` and `updateFunding` on the `PerpEngine` contract both assume a balanced book, where total long size equals total short size. `updatePositionFee` credits the liquidator `positionFeeDeltaUsd * (openInterest * 2)`, treating `openInterest` (longs only) as exactly half of total open size. `updateFunding` advances `cumulativeFundingUsd`, a transfer that nets to zero because what longs pay, shorts receive. A delist temporarily breaks that assumption: `settleDelistPosition` closes positions one at a time, so partway through a wind-down the open longs and shorts are unequal, and the product stays `ACTIVE` with no `delistInvoked` guard until `finalizeDelist`. On the contract in isolation, an update landing mid-wind-down would credit the liquidator on an `openInterest * 2` that no longer matches the paying size, or realize `fundingDeltaUsd * (remainingLongs - remainingShorts)` with no counterparty, minting withdrawable balance backed by nothing.

In the current system this state is not reachable due to the current sequencer implementation. Only the sequencer can submit action batches (`processActions` is `onlySequencer`), and it emits the whole delist under one engine message with funding and position-fee updates ordered before the settlement block, so any updates apply while the book is still full and balanced. Updates from an earlier message land before delisting starts; updates from a later message revert because the product is `DELISTED` by then (`updateFunding` / `updatePositionFee` require `status == ACTIVE`).

The contract does not enforce this ordering itself. Both functions gate only on `status == ACTIVE`, not on whether settlement has started, so the guarantee rests entirely on the sequencer's action ordering.

RECOMMENDATION

As a hardening measure, add a dedicated `delistSettlementStarted` flag, set by the first `settleDelistPosition`, and reject funding/fee updates only when that flag is true.

L-09 Updates During Delisting Mint Unbacked Balance

CONTINUED

— **RESOLUTION**

Status: Acknowledged.

LOW **L-10**

ACKNOWLEDGED

Partial Deleverage Blocks All Liquidations

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/758b8a8bc4ee7c54dc97809eac8a3d5d6f514836/src/Deleverage.sol#L498
REVIEW	Main Review

— DESCRIPTION

L-10 Partial Deleverage Blocks All Liquidations

CONTINUED

Auto-deleveraging unwinds a bankrupt singleton liquidator one quote-token book at a time. A sequence locks to a single quote token via `deleverageQuoteToken` and is split into batches that share one `messageIdx`. While `deleverageMessageIdx` is non-zero the sequence is in progress, and it is cleared in only one place: the `_finalizeDeleverageLiquidator` branch reached when the liquidator's last position closes. There is no abort or reset path.

Every batch, including that completion branch, re-proves the liquidator is bankrupt at the supplied price and reverts with `DeleverageLiquidatorNotBankrupt` otherwise. So if the liquidator's equity turns positive after a batch has closed part of the book but left a position open, every remaining batch and the completion branch revert, leaving `deleverageMessageIdx` set indefinitely.

Two things can lift the liquidator's equity mid-sequence. The mark price can recover between batches; or the new `MarginTransfer` action writes directly to the destination balance with no `deleverageMessageIdx` guard, so anyone can push margin into the liquidator subaccount and raise its `liqBalancePrior`. Because the liquidator is a singleton, one stuck sequence reverts `Liquidation.liquidateSubaccount` for every book via `LiquidatorIsDeleveraging`, freezing all liquidations across all quote tokens. Un-liquidatable accounts keep sliding, and any that reach negative equity become bad debt.

This is currently not reachable through the Nekuti engine or sequencer. The engine unwinds the liquidator in a single message at one frozen price snapshot, and every on-chain batch is a slice of that message embedding the same Pyth update, so the re-proof sees an identical price across batches and the mark cannot recover between them. The sequencer also relays actions strictly in creation order, so a `MarginTransfer` (not yet handled by the sequencer) from a later message is only submitted after the deleverage's batches complete and will not interleave.

It is still submitted because the contract enforces non-interleaving as an on-chain invariant.

`Liquidation.liquidateSubaccount` reverts `LiquidatorIsDeleveraging` with the [comment] (<https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/758b8a8bc4ee7c54dc97809eac8a3d5d6f514836/src/Liquidation.sol#L199-L200>) "that

a deleverage must complete as a batch and other actions may not interleave", and

`ExchangeConfig.updateLiquidator` is guarded identically.

— RECOMMENDATION

Two fixes are needed. First, guard any action that can credit the liquidator's balance in `deleverageQuoteToken` while a sequence is in progress. Revert with `LiquidatorIsDeleveraging` whenever `deleverageMessageIdx` is non-zero and the destination is the liquidator subaccount. Second, consider adding a sequencer or owner gated `abortDeleverage` action so a stuck sequence is recoverable without an upgrade. It should settle `deleverageBalanceDelta` under

L-10 Partial Deleverage Blocks All Liquidations

CONTINUED

the same accounting and solvency invariants as a normal completion, reset the delta to zero, and clear `deleverageMessageIdx` and `deleverageQuoteToken`.

— RESOLUTION

Status: Acknowledged.

LOW **L-11**

ACKNOWLEDGED

Timestamp Skew Causes Wrongful Liquidation

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/lib/LiquidationSharedLib.sol#L35-L84 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/Liquidation.sol#L220-L368
REVIEW	Main Review

DESCRIPTION

Liquidation and deleveraging value every product using a separate signed Pyth payload. Each price and timestamp can be authentic and accurate for when it was recorded. However, the protocol treats prices recorded at different times as they formed one coherent portfolio snapshot.

The parser extracts the timestamp and channel from each payload but requires the payload to contain only one feed:

```
``plain text
(data.timestamp, data.channel, feedsLength, pos) = PythLazerLib.parsePayloadHeader(payload);
```

```
if (feedsLength != 1) {
  revert PriceFeedsTooMany(id, feedsLength);
}
```

```
``
It then performs an individual stale price check and returns only the normalized price. The timestamp and channel are not returned to the liquidation or deleverage caller:
```

```
``plain text
if (block.timestamp - maxAge > data.timestamp) {
  revert PriceTooOld(id, data.timestamp, uint64(block.timestamp));
}
```

```
return uint128(data.price) * (10 ** exponent).toUint128();
``
```

Consequently, the caller cannot check whether the submitted prices have sufficiently close timestamps before aggregating them into one portfolio-equity calculation.

Consider an account holding a long position of 100 units in Product A and a short position of 100 units in correlated Product B, both opened at $\backslash\$100$. The account has $\backslash\$10,000$ of margin. At a coherent $\backslash\$100/\backslash\$100$ snapshot, neither position has PnL and the account retains $\backslash\$10,000$ of equity. At a coherent $\backslash\$200/\backslash\$200$ snapshot, the long gains $\backslash\$10,000$ while the short loses $\backslash\$10,000$, so the account still retains $\backslash\$10,000$ of equity.

Suppose Product A's signed payload reports $\backslash\$100$ at $t-59$, while Product B's payload

L-11 Timestamp Skew Causes Wrongful Liquidation**CONTINUED**

reports $\$200$ at t . With a configured 60-second maximum age, both payloads are individually recent enough. Combining them, however, gives the long zero PnL and the short a $\$10,000$ loss. The resulting synthetic portfolio has zero equity even though the account had $\$10,000$ of equity at both coherent snapshots.

The liquidation consequently succeeds and transfers the positions to the designated liquidator at the mixed prices. When the liquidator closes both positions at the coherent $\$200/\$200$ state, its inherited long realizes a $\$10,000$ gain equal to the victim's entire margin.

The same inconsistent valuation can falsely classify the system liquidator as bankrupt in both deleverage variants, causing solvent accounts to be wrongfully liquidated.

— RECOMMENDATION

Check the timestamp difference between every Pyth price used in the same liquidation or deleverage action and revert if it exceeds a configured limit.

— RESOLUTION

Status: Acknowledged.

LOW **L-12**

PENDING

Stale ADL Tolerance Can Lock Liquidations

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/Deleverage.sol#L397-L451 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/DeleverageWithCompensation.sol#L423-L477
REVIEW	Main Review

— DESCRIPTION

A complete ADL sequence is intended to close all of the liquidator's positions. However, the sequence may be divided into multiple on-chain actions because of match or transaction limits. Intermediate actions can therefore close some positions while leaving the remaining positions for subsequent actions.

Both deleverage implementations calculate bankruptcy tolerance from every liquidator position present at the beginning of each action. When an intermediate action closes some positions, its final validation still uses the original tolerance, including contributions from positions that were already closed.

This allows the intermediate action to commit when the liquidator's ending equity exceeds the tolerance calculated from its remaining positions but remains within the original tolerance. The next action recalculates tolerance from only the remaining positions and reverts because the liquidator is now considered solvent.

Engine-generated ADLs are expected to settle the liquidator at its bankruptcy price, and tested engine sequences did not produce positive residual equity. Triggering the issue therefore requires an accepted engine/on-chain discrepancy that leaves positive equity within the stale-tolerance range.

If this occurs, the retained deleverage marker prevents the sequence from completing, blocks subsequent liquidations, and prevents the owner from rotating the liquidator.

— RECOMMENDATION

After processing each intermediate action, recalculate bankruptcy tolerance using only the liquidator positions that remain open. Reject the action if its ending equity exceeds the recalculated tolerance.

— RESOLUTION

Status: Pending.

LOW **L-13**

PENDING

Deferred PnL Drift Can Freeze ADL

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/Deleverage.sol#L683-L700 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/Deleverage.sol#L416-L449 https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eeb953574540e4a10b8e0ad4f1e/src/Deleverage.sol#L33-L338
REVIEW	Main Review

— DESCRIPTION

ADL accepts differences between the engine-provided realized PnL and the value recomputed on-chain when each difference is within the configured tolerance. For incomplete chunks, these values are stored in `deleverageBalanceDelta`, while the post-chunk bankruptcy check uses mark price deltas that may omit discrepancies from closed positions.

The same divergence can occur without any realized PnL discrepancy. Preparation floors the whole position mark notional once, while each match independently floors its mark-notional delta. Fragmented reductions can therefore cause the cumulative mark-PnL delta to understate the equity represented by the deferred realized PnL and the remaining position cost.

If either type of discrepancy accumulates, a chunk can commit even though the liquidator's ending equity exceeds the tolerance supported by its remaining positions. The next continuation includes the deferred balance delta and recalculates equity and tolerance from the remaining positions, causing it to revert because the liquidator appears solvent.

The sequencer verifies each PnL difference separately, so multiple allowed differences can accumulate. If a later ADL chunk fails during simulation, the relayer can still submit the earlier chunks that passed, leaving an incomplete sequence on-chain.

A failed continuation leaves the deleverage marker set, blocking subsequent liquidations and liquidator replacement.

— RECOMMENDATION

For incomplete chunks, calculate post-chunk bankruptcy equity from the exact realized PnL, funding, and fees that will be persisted, plus the mark PnL of the remaining positions. Alternatively, explicitly carry every accepted discrepancy into the post-chunk equity calculation.

As defense in depth, the sequencer should reproduce the contract's accounting across the complete ADL sequence and reject the sequence if cumulative discrepancies would cause any continuation to fail

L-13 Deferred PnL Drift Can Freeze ADL

CONTINUED

— **RESOLUTION**

Status: Pending.

LOW **L-14**

ACKNOWLEDGED

Token Removal Ignores Pending Withdrawals

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/ExchangeConfig.sol#L270-L276
REVIEW	Main Review

— DESCRIPTION

Token removal is prevented when a token has an active balance or pending deposit, but pending withdrawals are not included in this validation. Because withdrawal initiation removes the amount from `totalBalance` and `totalBackedBalance`, the owner can remove a token after its final active balance has been converted into a pending withdrawal.

The removal check only considers active balances and pending deposits. Removing the token deletes its address mapping and configuration, including the OFT address required by cross-chain withdrawal finalization.

Pending withdrawals may consequently remain unclaimable until the token configuration is restored.

— RECOMMENDATION

Consider tracking the aggregate pending withdrawal amount for each token and require it to be zero before allowing token removal.

— RESOLUTION

Status: Acknowledged.

LOW **L-15****PENDING**

Missed intra-user equity peaks can indefinitely disable all liquidations while the backstop is solvent

CATEGORY	Informational
LOCATION	<code>packages/contracts-team1-1782406004832/src/Deleverage.sol:deleverageLiquidator():L251-L280</code> <code>packages/contracts-team1-1782406004832/src/Deleverage.sol:_processDeleverageUser():L533-L609</code> <code>packages/contracts-team1-1782406004832/src/Deleverage.sol:_finalizeDeleverageLiquidator():L397-L453</code> <code>packages/contracts-team1-1782406004832/src/DeleverageWithCompensation.sol:deleverageLiquidatorWithCompensation():L275-L307</code> <code>packages/contracts-team1-1782406004832/src/DeleverageWithCompensation.sol:_processDeleverageUserChunk():L558-L605</code> <code>packages/contracts-team1-1782406004832/src/Liquidation.sol:liquidateSubaccount():L202-L209</code> <code>packages/contracts-team1-1782406004832/src/ExchangeConfig.sol:updateLiquidator():L436-L450</code>
REVIEW	Main Review

— DESCRIPTION

Each match computes a liquidator unrealized-mark-PnL delta, but both deleverage modules aggregate all matches for one user/chunk before updating the maximum cumulative delta. Consequently, an early match in a normal canonical product traversal can restore positive equity while a later match erases that peak before the explicitly documented between-match bankruptcy invariant samples it. The PoC starts at $-\$20$, processes $+\$50$ and then $-\$30$, and commits at zero even though intermediate equity reached $+\$30$; splitting identical matches across user entries correctly reverts.

Impact: The invalid partial action persists `deleverageMessageIdx`. If a remaining position becomes profitable, continuations reject the now-solvent liquidator, while all unrelated liquidations and liquidator replacement remain blocked. Fairly closing the position merely realizes the surplus into cash and still cannot clear the lock. The global liquidation halt can persist indefinitely unless markets reverse, the protocol sacrifices positive backstop value, or it upgrades, allowing bad debt to accumulate.

— RECOMMENDATION

Update cumulative and maximum liquidator mark-PnL deltas after every individual match in both deleverage modules. Add canonical ascending-product multi-match regressions and a constrained stale-lock recovery path that explicitly accounts for deferred PnL and positive surplus.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-01

ACKNOWLEDGED

Liquidations Always Seize A User's Entire Margin

CATEGORY	Suggestion
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/Liquidation.sol#L370-L387
REVIEW	Main Review

DESCRIPTION

A subaccount is liquidatable once its equity drops below its maintenance margin, $MM = \text{notional} * (0.5 / \text{maxLeverage} + \text{takerFee})$, roughly 5% of notional at 10x. When `Liquidation.liquidateSubaccount` runs it transfers both the position and the account's remaining margin to the liquidator: it credits `userEquityAfterLiquidation` to the liquidator via `balanceTransfer` and sets `userAccount.balance[subaccount][quoteToken] = 0`. The liquidated user is left with nothing. The position could instead be matched against a resting order and closed on the book. In that case the user pays only the market-close cost (slippage plus taker fee) and keeps the rest of their margin after being liquidated.

RECOMMENDATION

Consider attempting to close a liquidatable position by matching it against resting orders. If book liquidity is sufficient to fill the position, close it there and leave the residual margin to the user. Only when liquidity is insufficient should the position and its remaining margin be transferred to the liquidator as a fallback.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-02

RESOLVED

Signed Oracle Prices Decode as Unsigned

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/lib/LiquidationSharedLib.sol#L57-L64
REVIEW	Main Review

DESCRIPTION

Pyth Lazer encodes `Price`, `BestBidPrice`, and `BestAskPrice` as signed 64-bit integers. Meridian instead decodes these properties using an unsigned reader and stores the operative price as `uint64`.

Consequently, a negative mantissa is reinterpreted as a large unsigned integer. For example, `-1` becomes `264 - 1` before exponent normalization, producing an extremely large positive price.

The sequencer rejects nonpositive prices before caching them, substantially mitigating the issue. However, if an authenticated negative update is unexpectedly relayed, delisting can store an incorrect expiry price, while liquidation and deleveraging can make incorrect eligibility calculations or revert. `BestBidPrice` and `BestAskPrice` have the same type mismatch but are currently discarded.

RECOMMENDATION

Decode the affected properties as `int64`, require the operative price to be present and greater than zero, and only then cast and normalize it.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/356>

INFORMATIONAL I-03

RESOLVED

Missing View Modifiers in Gateway Interface

CATEGORY Informational

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/8edb3d1e8f7b933e51d13ef84cbc3cf06695bc0d/src/interface/IXchangeGateway.sol#L139-L145>

REVIEW Main Review

DESCRIPTION

The IXchangeGateway interface omits the view modifier from two view getter declarations, although their implementations are correctly marked view.

```
``plain text
function getAccountGlobalTotalBalance(address token) external returns (uint256);
function getAccountGlobalTotalPendingBalance(address token) external returns (uint256);
``
```

Consequently, Solidity integrations cannot call them from their own view functions through this interface, and clients generated from its ABI may incorrectly expose them as transaction methods.

RECOMMENDATION

Add the view modifier to both interface declarations.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/355>

INFORMATIONAL **I-04**

RESOLVED

Duplicate Using Directives In PerpEngine

CATEGORY Informational

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L79-L86>

REVIEW Main Review

DESCRIPTION

The `PerpEngine` contract declares the same attached-library directive twice for three libraries: `using Exchange for Exchange.Data` at lines 79 and 84, `using PerpPosition for PerpPosition.Data` at lines 80 and 86, and `using FeeCollector for FeeCollector.Data` at lines 81 and 85.

RECOMMENDATION

Delete the duplicate declarations at lines 84, 85 and 86, keeping the originals at lines 79, 80 and 81.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/355>

INFORMATIONAL I-05

ACKNOWLEDGED

Unsupported Exponents Halt Risk Actions

CATEGORY	Suggestion
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/lib/LiquidationSharedLib.sol#L72-L84
REVIEW	Main Review

DESCRIPTION

The protocol only accepts Pyth exponents between -8 and -1 and normalizes prices to D9 through multiplication.

```
``plain text
if (data.exponent < MIN_EXPONENT || data.exponent > MAX_EXPONENT) {
    revert PriceInvalidExponent(id, data.exponent);
}
uint16 exponent = uint16(9 + data.exponent);
``
```

If a configured Pyth feed is rescaled to -9 or lower under the same feed ID, its authenticated payload becomes unusable. An account holding the affected product cannot be liquidated: including it causes price validation to revert, while excluding it violates the requirement to process all positions in the same collateral pool. The unsupported price can also prevent delist initiation and ADL involving that product. The off-chain sequencer rejects out-of-range updates before caching them, so unsupported payloads do not normally reach the contract. If no valid supported price remains available, affected risk actions remain blocked until the owner changes the feed or upgrades the protocol.

RECOMMENDATION

Monitor configured Pyth feed exponents and update the affected feed ID before any announced rescaling becomes effective.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-06

RESOLVED

Packed ADL Fields Limit Settlements

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0e9b953574540e4a10b8e0ad4f1e/src/interface/IDeLeverage.sol#L52-L64
REVIEW	Main Review

DESCRIPTION

DeLeverage actions minimize calldata by encoding each match with 64-bit fields, while positions and the subsequent processing structure use 128-bit values.

At D9 precision, uint64 can encode approximately $\$18.45$ billion, while int64 can encode PnL between approximately $-\$9.22$ billion and $+\$9.22$ billion. Internally, significantly larger position costs remain representable.

Consider a liquidator holding a one raw D9 unit long position with a cost of approximately $\$27.67$ billion. Closing a long calculates its PnL as the submitted notional minus its stored cost. Even using the maximum uint64 notional produces expected PnL below int64.min. The closest encodable PnL differs from the expected value by 2,220,449 raw units, exceeding the 2,220,448-unit tolerance by one raw unit. Therefore, no packed ADL match can close the position.

The extreme position therefore cannot be settled through the packed ADL path. However, ordinary order matching uses wider fields and can close it, and the required position is far outside current configuration limits, so this does not create a protocol lock.

RECOMMENDATION

Document the ADL format's 64-bit per-match limits and clarify that products capable of producing unrepresentable minimum-size settlements fall outside the supported configuration range.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/sequencer/commit/be5c7238b3c4b4dfcfef02e18e4215bd0f245fa1>

INFORMATIONAL I-07

PENDING

Position Fee Rebates Are Not Supported

CATEGORY	Suggestion
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0e9b953574540e4a10b8e0ad4f1e/src/PerpEngine.sol#L311-L341
REVIEW	Main Review

DESCRIPTION

The sequencer accrues position fees by calling `updatePositionFee` on the `PerpEngine` contract, which advances the per-product `cumulativePositionFeeUsd` index and credits the liquidator for the fee owed by both sides of the open interest. The matching engine supports negative position fees as rebates, where the beneficiary account instead pays out to position holders in proportion to their absolute position size. The contract cannot represent this. `positionFeeDeltaUsd` is a `uint128` applied as `product.cumulativePositionFeeUsd + positionFeeDeltaUsd`, and the liquidator credit is applied with `+=`, so both the index and the beneficiary balance can only ever increase.

RECOMMENDATION

Consider documenting that rebates are intentionally unsupported in this deployment. Otherwise, change `positionFeeDeltaUsd` and `PerpProduct.cumulativePositionFeeUsd` to `int128` to mirror `cumulativeFundingUsd`, update the realized-fee arithmetic in `PerpEngine`, `Liquidation` and `Deleverage` to signed math, and apply the liquidator delta through `LiquidationSharedLib.updateLiquidatorBalance`, which already reverts on a negative resulting balance and so reproduces the beneficiary margin check the engine performs before applying a rebate round.

RESOLUTION

Status: Pending.

INFORMATIONAL I-08

RESOLVED

Zero Max Quantity Renders A Product Close-Only

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/PerpEngineAdmin.sol#L228-L230>

REVIEW Main Review

DESCRIPTION

The `_verifyProductSizes` function from the `PerpEngineAdmin` contract rejects a zero `tickSize` and a zero `lotSize`, but its `maxQuantity % lotSize != 0` guard passes for a zero `maxQuantity` because `0 % lotSize == 0`, so both `register` and `updateProduct` accept a `maxQuantity` of zero.

Every order on such a product then fails the `order.quantity > product.maxQuantity` check in `PerpEngine`, leaving only zero-quantity reduce-only orders valid. So existing holders can still exit against each other, but no position can be opened or increased until an owner update restores a non-zero value.

RECOMMENDATION

Add `if (maxQuantity == 0) revert InvalidParameter("maxQuantity", P_ERR_AMT_ZERO);` to `_verifyProductSizes`

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/359>

INFORMATIONAL I-09

ACKNOWLEDGED

Disabling The Mark Price Cap Is Dangerous

CATEGORY	Informational
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/lib/storage/Account.sol#L48
REVIEW	Main Review

DESCRIPTION

The engine's cap behaviour is supplied per mark update and the sequencer currently hardcodes `CapToFirstBankruptcyPrice`, which pins every price move at the point where the first account reaches exactly zero equity. No account can therefore be pushed to negative equity by a price move, which is what guarantees a liquidated account is zeroed out with `balanceTransfer = 0` and leaves no deficit for the liquidator to absorb.

Under `NoCap` a large price jump can leave an account at negative equity, producing a negative `balanceTransfer`, and because balances are unsigned the liquidator cannot carry that as debt: `updateLiquidatorBalance` from the `LiquidationSharedLib` library reverts with `LiquidatorInsufficientBalance` if the combined realized PnL and balance transfer would take its quote token balance below zero.

The account is then unliquidatable until the liquidator is funded with enough cash, leaving the position open to sink further into bad debt.

RECOMMENDATION

Document that keeping `CapToFirstBankruptcyPrice` enabled is critical for liquidation liveness.

If `NoCap` is ever adopted for a product, account balances need redesigning so they can be signed and hold a negative value where required, and the settlement path updated to carry that deficit rather than revert on it.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-10

ACKNOWLEDGED

Emergency Pause Blocks Liquidations

CATEGORY	Warning
LOCATION	https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/ExchangeGateway.sol#L337-L343
REVIEW	Main Review

— DESCRIPTION

The `emergencyPause` function from the `ExchangeGateway` contract sets a single global flag that gates `processActions` alongside the deposit and withdraw entrypoints, so a pause halts every sequencer-relayed action, including liquidations and reduce-only closes, as well as user collateral top-ups. Because mark prices keep moving while the book is frozen, a position that was safely liquidatable when the pause began can be deeply insolvent by the time trading resumes.

— RECOMMENDATION

Pausing the liquidation and deposit paths remains important, since pause is invoked when an exploit is suspected and those functions may themselves sit in the exploit path, so no code change is recommended here.

Consider documenting that risk-reducing operations being included in the paused set of operations is an intended tradeoff

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-11

PENDING

Zero-valued withdrawal request ID conflicts with the fresh Claimer cursor

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/lib/storage/ClaimerWithdrawal.sol:load():L9-L20 packages/contracts-team1-1782406004832/src/Claimer.sol:finalizeWithdrawals():L79-L95
REVIEW	Main Review

DESCRIPTION

The Claimer's sole high-water mark initializes to zero and every submitted ID must be strictly greater than it. Therefore ID zero can never be processed. Repository tests consistently start fresh streams at ID 1, but neither the interface nor the type explicitly reserves zero, creating an integration ambiguity if an external service uses zero-based numbering.

Impact: If the intended external stream starts at zero, its genesis request and any batch retaining it cannot use the Claimer until the operator omits or rennumbers it. The withdrawal remains directly finalizable through the Gateway, so this is avoidable integration friction rather than loss or protocol-level denial.

RECOMMENDATION

Explicitly reserve and document zero and reject it clearly, or add an initialized flag so a valid first ID of zero can be distinguished from an uninitialized cursor.

RESOLUTION

Status: Pending.

INFORMATIONAL I-12

RESOLVED

Token configuration NatSpec incorrectly labels D9 values as native units

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/interface/IXExchangeConfig.sol:updateToken():L108-L111 packages/contracts-team1-1782406004832/src/ExchangeConfig.sol:updateToken():L210-L248
REVIEW	Main Review

— DESCRIPTION

The implementation, deployment scripts, and tests consistently interpret depositCapacity, minDeposit, depositFee, and withdrawFee as normalized D9 accounting values. However, UpdateTokenParams and Token.Data NatSpec describe several or all of these fields as token-native units. The original scale-distortion claim therefore requires trusted-owner misconfiguration and is not an exploitable accounting bug, but the public API documentation is contradictory.

Impact: An operator or integrator following the stale NatSpec instead of established deployment conventions could supply incorrectly scaled configuration for non-nine-decimal collateral. Because token configuration is a fully trusted owner action, this is documentation and operational ambiguity rather than a permissionless vulnerability.

— RECOMMENDATION

Update IXExchangeConfig.UpdateTokenParams and Token.Data NatSpec to state that all capacity, minimum, and flat-fee fields use D9 precision. Consider encoding the unit in field names or dedicated value types and adding decimal-diverse configuration tests.

— RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/359>

INFORMATIONAL I-13

PENDING

Zero-normalized deposits register collateral-free subaccounts

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/CollateralManager.sol:_verifyAndDeposit():L409-L508 packages/contracts-team1-1782406004832/src/share/tokens/ERC20Helpers.sol:normalize():L10-L19 packages/contracts-team1-1782406004832/src/ActionHandler.sol:_handleFinalizeDeposit():L321-L354
REVIEW	Main Review

— DESCRIPTION

For a deposit-enabled token with more than nine decimals and owner-configured minDeposit and depositFee of zero, a positive native amount below one D9 accounting unit normalizes to zero. The deposit path accepts it, stores a pending deposit identified by a nonzero token address, and transfers zero tokens. A sequencer can then finalize the zero pending amount and mark the named subaccount registered without collateral.

Impact: An attacker paying transaction gas can create persistent zero-value pending deposits and, if the sequencer finalizes them, zero-balance registered subaccounts. No collateral, withdrawal capacity, or cross-user value is created, and no bounded onchain iteration is affected, so the proven impact is limited to state cleanliness and avoidable indexing/finalization overhead under a zero-minimum configuration.

— RECOMMENDATION

Reject deposits when normalizedAmount or normalizedNetAmount is zero. Also consider requiring a positive minimum deposit for every deposit-enabled concrete token and defensively rejecting zero-total pending deposits during finalization.

— RESOLUTION

Status: Pending.

INFORMATIONAL I - 14

ACKNOWLEDGED

Expired partial-order fills are never pruned

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/PerpEngine.sol:_settleOrderFills():L800-L844 packages/contracts-team1-1782406004832/src/ActionHandler.sol:_handleMatchOrders():L434-L456
REVIEW	Main Review

DESCRIPTION

_settleOrderFills stores `product.filled\[digest\]` for a partially filled order and deletes it only if that exact digest is later fully filled. Once the signed order expires, ActionHandler rejects any further match before the deletion branch, and the protocol exposes no cancellation or expiry-pruning path. Consequently, the partial-fill slot remains in proxy storage indefinitely.

Impact: Expired partial orders leave stale proxy/L2 state that cannot be reclaimed. Under ordinary honest minimum-fill matching, however, each such slot is paired with a consumed nonce for the fully filled counter-order and does not demonstrate asymptotic storage amplification beyond normal permanent replay bookkeeping. The stronger two-slot/no-nonce construction requires trusted-sequencer underfilling and a directionally invalid reduce-only self-order, so no Low-severity unprivileged impact is established.

RECOMMENDATION

Provide authenticated cancellation or expiry-based pruning for `filled\[digest\]`, with the signed expiry cryptographically bound to the digest. Consider similarly pruning expired replay state or using bounded epochs so obsolete order bookkeeping can be reclaimed safely.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-15

PENDING

Revoked linked signer addresses cannot later become trader accounts

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/lib/storage/Account.sol:revokeSigner() :L140-L158 packages/contracts-team1-1782406004832/src/CollateralManager.sol:_verifyAndDeposit() :L473-L487
REVIEW	Main Review

DESCRIPTION

Revocation permanently marks the former signer in `excludedAccounts`, so the address cannot subsequently initialize an exchange account through a deposit. This behavior is deliberate and covered by upstream tests, but the irreversible account-address consequence is broader than merely preventing signer reuse and should be made explicit to users when they co-sign a link.

Impact: A consenting linked-signer address that is later revoked must use a different wallet if its owner wants to become an independent Meridian trader. No third-party funds or protocol invariant are affected.

RECOMMENDATION

Prominently disclose during signer linking that the address is permanently reserved from use as an exchange account, or redesign identity state if future reuse as an independent account is desired.

RESOLUTION

Status: Pending.

INFORMATIONAL I-16

PENDING

Synthetic token capacity is accepted but has no defined effect

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/ActionHandler.sol:_handleConvertToken() :L555-L613 packages/contracts-team1-1782406004832/src/ExchangeConfig.sol:updateToken() :L210-L249
REVIEW	Main Review

DESCRIPTION

The shared token update API accepts and reports a nonzero `depositCapacity` for synthetic tokens, even though synthetic tokens cannot enable physical deposits and signed conversions do not consult the destination token's capacity. Consequently, the configured value has no effect on balances created by conversion. The behavior is not a demonstrated risk-limit bypass because the field is documented as a deposit restriction and conversion preserves the backing family's already-counted exposure, but accepting an inapplicable value creates configuration/API ambiguity.

Impact: An operator or integrator may incorrectly interpret a stored synthetic-token capacity as an enforced internal exposure cap. No collateral is minted, backing-family capacity is not bypassed, and the PoC shows no direct third-party loss.

RECOMMENDATION

Either reject or ignore nonzero deposit-only settings for synthetic tokens and document them as inapplicable, or introduce a separate explicitly defined conversion/exposure capacity and enforce it when crediting synthetic destinations.

RESOLUTION

Status: Pending.

INFORMATIONAL I-17

PENDING

Per-account caps cannot prevent narrow ADL recovery from freezing all liquidations

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/lib/ActionDecoder.sol:decodeDeleverageLiquidatorAction():L287-L325 packages/contracts-team1-1782406004832/src/Deleverage.sol:_prepareDeleverageLiquidator():L315-L395 packages/contracts-team1-1782406004832/src/Deleverage.sol:_finalizeDeleverageLiquidator():L397-L454 packages/contracts-team1-1782406004832/src/Deleverage.sol:_processDeleverageUser():L532-L567 packages/contracts-team1-1782406004832/src/Liquidation.sol:liquidateSubaccount():L195-L219 packages/contracts-team1-1782406004832/src/Liquidation.sol:_settleLiquidatorPositionTakeover():L707-L800 packages/contracts-team1-1782406004832/src/ExchangeConfig.sol:updateLiquidator():L435-L451
REVIEW	Main Review

DESCRIPTION

Per-subaccount offchain caps do not bound the designated liquidator's aggregate exposure: each full int128 liquidation nets another user's position into its existing position without maxQuantity, maxPositionNotionalUsd, or an int64-compatible aggregate check. The revised public-path PoC proves three separately funded and signed int64-capped accounts aggregate through genuine liquidations into a $3 \times \text{int64.max}$ liquidator position. Repeating the same cap-compliant operation can produce the $100,001 \times \text{int64.max}$ exposure while remaining within the configured product OI/notional limits. Because packed ADL settlements carry only int64 size and related 64-bit fields, a complete unwind then needs 100,001 settlements and a 3.4MB payload, exceeding Nitro's transaction-data bound and failing Robinhood's 32m gas cap. A bounded partial ADL necessarily stores the singleton marker; fresh favorable pricing makes the residual solvent and continuation reverts, while no solvent-abort path clears the marker.

Impact: An honest liquidation cascade across individually cap-compliant accounts can force the backstop unwind across transactions. Favorable price movement can then leave a solvent residual that cannot be further ADL'd or normally aborted, while the singleton marker blocks unrelated liquidations across all products and quote pools and prevents liquidator rotation. Underwater accounts can remain unresolved and bad debt can grow until prices reverse or governance deploys an emergency upgrade. Even without recovery, narrow recovery fields can cause a prolonged liquidation outage during a large cascade.

RECOMMENDATION

Apply aggregate quantity and notional limits to liquidator takeovers that are compatible with ADL wire and production transaction bounds, and widen ADL size/notional/PnL fields to the position/liquidation int128/uint128 domain. Add a solvent-abort transition that reconciles deferred balance state and clears the marker without force-closing counterparties, and scope non-interleaving more narrowly where possible.

I-17 Per-account caps cannot prevent narrow ADL recovery from freezing all liquidations **CONTINUED**

— **RESOLUTION**

Status: Pending.

INFORMATIONAL I-18

ACKNOWLEDGED

Local withdrawals silently discard low-decimal collateral remainders created by normal trading

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/ActionHandler.sol:_handleInitiateWithdraw():L286-L305 packages/contracts-team1-1782406004832/src/CollateralManager.sol:_completeWithdrawal():L293-L307 packages/contracts-team1-1782406004832/src/sha-re/tokens/ERC20Helpers.sol:denormalize():L21-L30
REVIEW	Main Review

— DESCRIPTION

Meridian accounts for collateral at nine decimals, but denormalization of tokens with fewer decimals uses truncating division. Withdrawal initiation debits the complete normalized payout and reduces global accounting by that amount, whereas same-chain non-USD finalization transfers only the truncated native-token amount, clears the pending request, and does not re-credit the remainder. The updated PoC demonstrates that ordinary trading fees can produce such a nonrepresentable balance for standard six-decimal USDC, then closes all positions and withdraws the full balance through the local path. Cross-chain OFT flooring and rescue behavior are excluded as already covered by the prior finding.

Impact: A local withdrawal of low-decimal collateral silently loses any sub-native-unit remainder created by trading fees, PnL, funding, or similar accounting. Each loss is strictly less than one smallest native token unit, but the residual tokens accumulate untracked in the Gateway while the user's internal and pending balances are cleared.

— RECOMMENDATION

Before debiting a local concrete-token withdrawal, require its net payout to be exactly representable in the token's native decimals. Alternatively, transfer the floored amount while retaining or re-crediting the normalized remainder with consistent global accounting. Where feasible, align fees and other balance-producing calculations to the quote token's native quantum.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-19

RESOLVED

Deposit Capacity Behavior Is Undocumented

CATEGORY Documentation

LOCATION <https://github.com/GuardianOrg/contracts-team1-1782406004832/blob/4fb89b928a3a0eb953574540e4a10b8e0ad4f1e/src/CollateralManager.sol#L461-L470>

REVIEW Main Review

DESCRIPTION

The documentation does not clearly state that `depositCapacity` limits active backed balances rather than the exchange's physical token custody. Initiated withdrawals are removed from active accounting immediately and therefore release deposit capacity, although their tokens remain escrowed until finalization or rescue.

Consequently, physical custody may exceed `depositCapacity` by the aggregate pending withdrawal amount. This does not create insolvency because each withdrawal claim remains fully backed, but the undocumented distinction may cause the parameter to be interpreted as a total custody limit.

RECOMMENDATION

Document that `depositCapacity` excludes initiated withdrawals and limits only active backed collateral.

RESOLUTION

Status: Resolved.

<https://github.com/meridianxyz/contracts/pull/359>

INFORMATIONAL I-20

ACKNOWLEDGED

Pending withdrawals bypass the token-specific completion halt

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/ActionHandler.sol:_handleInitiateWithdraw():L273-L305 packages/contracts-team1-1782406004832/src/CollateralManager.sol:finalizeWithdraw():L240-L252 packages/contracts-team1-1782406004832/src/CollateralManager.sol:_completeWithdrawal():L280-L307
REVIEW	Main Review

DESCRIPTION

Token.withdrawEnabled is checked only when a withdrawal is initiated. Although the README says disabling the onchain flag halts the relayer and prevents pending-withdrawal completion, finalizeWithdraw does not reload Token.Data, allowing an account to directly complete a pre-existing request while the token remains disabled.

Impact: If the flag is intended as a token-scoped completion halt, operators cannot reliably stop custody transfers for one affected token: queued requests remain directly executable, while the only effective onchain alternative is the global emergency pause. No accounting or solvency loss is proven because initiation already debits the payout from backed liabilities.

RECOMMENDATION

Clarify the intended semantics. If disabling withdrawals must stop pending transfers, re-check withdrawEnabled before local and OFT completion, with any rescue override explicitly scoped. Otherwise, document the flag as initiation-only and align relayer behavior so it does not imply a token-scoped completion guarantee.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-21

PENDING

Binary64 tolerance can admit liquidation marginally above exact maintenance

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/Liquidation.sol:liquidateSubaccount():L275-L305 packages/contracts-team1-1782406004832/src/Liquidation.sol:liquidateSubaccount():L356-L387 packages/contracts-team1-1782406004832/src/Liquidation.sol:_processLiquidateSubaccountWithCompensationProduct():L574-L635 packages/contracts-team1-1782406004832/src/Liquidation.sol:_verifyMaintenanceMargin():L831-L849
REVIEW	Main Review

DESCRIPTION

Liquidation accepts an offchain eligibility classification within a cumulative per-position IEEE-754 tolerance. The PoC models round-to-nearest-even binary64 arithmetic for a correctly margined, production-capped three-product portfolio: the model reports slightly below maintenance while Solidity's exact calculation remains 20 raw D9 units above it, and the accumulated onchain tolerance admits liquidation. The production matching-engine source and its precise conversion order are not available to confirm this modeled discrepancy.

Impact: A portfolio deliberately or coincidentally positioned within a few raw D9 units of maintenance may be liquidated despite remaining marginally above the unbuffered Solidity threshold. Because the protocol intentionally zeroes an entire isolated book after eligibility, the transition has a large cliff effect, but the discrepancy is nanodollar-scale, requires an extremely precise boundary state, and is within the documented rounding accommodation.

RECOMMENDATION

Use fixed-point offchain risk arithmetic matching the canonical onchain calculation. If binary64 remains necessary, document and formally derive a portfolio-wide error bound from the production engine's exact evaluation order, and decide explicitly which calculation is authoritative at the liquidation boundary.

RESOLUTION

Status: Pending.

INFORMATIONAL I-22

PENDING

Tolerated IEEE-754 notional drift pollutes position basis

CATEGORY	Informational
LOCATION	packages/contracts-team1-1782406004832/src/PerpEngine.sol:_verifyFillNotional():L594-L603 packages/contracts-team1-1782406004832/src/PerpEngine.sol:_settlePosition():L629-L710 packages/contracts-team1-1782406004832/src/PerpEngine.sol:_executeMatch():L862-L893
REVIEW	Main Review

DESCRIPTION

The contract derives quantity times the maker's signed price but uses it only as a tolerance reference; a differing raw-D9 float64 matching-engine fillNotional within `(reference >> 52) + 13` is stored verbatim as position cost. The passing PoC uses a \$14.42m position fully collateralized for configured 10x leverage and charges fixture maker/taker fees. The raw float64 engine notional exceeds canonical Solidity arithmetic by two units. Forty-eight same-price open/close cycles leave the attacker exactly 96 raw units above its fees-only baseline and the victim 96 below, after which the resulting balance is finalized into backing USD. Impact: Accepted offchain floating-point dust becomes a real zero-sum trader-to-trader accounting transfer rather than validation-only tolerance. It pollutes signed-price position basis and downstream realized PnL even with sufficient margin and configured fees. The per-fill value is tiny and normal fees dominate it, supporting Low severity.

RECOMMENDATION

Use canonical `fillQuantity.mulDecimal(makerPrice)` for every position-cost mutation and downstream PnL calculation, retaining external fillNotional only as a validation input. Alternatively canonicalize the engine value to Solidity D9 arithmetic and require exact equality.

RESOLUTION

Status: Pending.

§ 05

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.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.