PERP CITY · SECURITY REVIEW

Perp City
PerpCity Contracts Updates.

A security review of PerpCity Contracts Updates.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Perp City	PerpCity Contracts Updates	Zdravko Hristov, Andreas Zottl, Ferran Celades, Vlad Zotov, Michael Lett	October 19 – November 9, 2026	November 27, 2026

Published at guardian.security/portfolio/perp-city/perpcity-contracts-updates-2026-11-27

§ 00

Contents.

55 pages: overview, confidence ranking, scope, 29 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. 08

Main Review · 26 findings P. 10

Remediation Review · 3 findings P. 48

§ 05 **Disclaimer** P. 54

§ 06 **About Guardian** P. 55

§ 01

Overview.

Guardian reviewed PerpCity Contracts Updates across 2 review rounds. The final report includes 7 Low, 22 Informational findings.

AUDIT FIRM
Guardian

CLIENT
Perp City

PROJECT
PerpCity Contracts Updates

PREPARED BY
Zdravko Hristov, Andreas Zottl, Ferran Celades, Vlad Zotov, Michael Lett

AUDIT WINDOW
October 19 – November 9, 2026

FINAL REPORT
November 27, 2026

REPOSITORY
`https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts`

METHODOLOGY
Smart contract · Manual · Main Review · Remediation Review

MAIN REVIEW COMMIT
`915821ace0cd`

REMEDIATION REVIEW COMMIT
`6e830949c5e3`

1.1 Confidence Ranking

5 / 5

Very High Confidence

Guardian assigns a Confidence Ranking of 5 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	**0**	0	0	0	0
Medium	**0**	0	0	0	0
Low	**7**	0	3	0	4
Informational	**22**	0	10	0	12
All severities	**29**	**0**	**13**	**0**	**16**

§ 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 ASSIGNED	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	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 PerpCity Contracts Updates in `https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts`. The scope includes 3,612 source lines across 26 files.

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 1/2.

FILE	NSLOC	TOTAL
MAIN REVIEW		
src/AccountingToken.sol	5	84
src/interfaces/IPerp.sol	197	294
src/interfaces/IWithdrawalController.sol	19	47
src/libraries/Constants.sol	4	58
src/libraries/Errors.sol	53	67
src/libraries/PerpAmmLogic.sol	302	429
src/libraries/PerpFundingLogic.sol	40	76
src/libraries/PerpLiquidationLogic.sol	245	427
src/libraries/PerpLogic.sol	468	959
src/libraries/PerpMakerLogic.sol	331	413
src/libraries/PerpPreviewLogic.sol	49	102
src/libraries/PerpUtilizationLogic.sol	202	253
src/libraries/SafeBeaconIndex.sol	16	28
src/libraries/SignedFixedPointMathLib.sol	18	69
src/libraries/Structs.sol	161	447
src/ModuleRegistry.sol	4	36
src/modules/Fees.sol	48	260
src/modules/Funding.sol	25	89
src/modules/MarginRatios.sol	93	191
src/modules/PriceImpact.sol	7	72
src/modules/Pricing.sol	27	78
src/Perp.sol	346	713
src/PerpFactory.sol	76	171
src/PerpGuardHook.sol	10	102
src/ProtocolFeeManager.sol	13	46
src/WithdrawalController.sol	144	254
REMEDIATION REVIEW		
src/AccountingToken.sol	7	66
src/interfaces/IPerp.sol	28	846
src/interfaces/IWithdrawalController.sol	30	181
src/libraries/Constants.sol	10	72
src/libraries/Errors.sol	0	168
src/libraries/PerpAmmLogic.sol	75	434
src/libraries/PerpFundingLogic.sol	23	78
src/libraries/PerpLiquidationLogic.sol	78	415

Files in Scope 2/2.

FILE	NSLOC	TOTAL
src/libraries/PerpLogic.sol	156	781
src/libraries/PerpMakerLogic.sol	45	417
src/libraries/PerpPreviewLogic.sol	15	93
src/libraries/PerpUtilizationLogic.sol	52	261
src/libraries/SafeBeaconIndex.sol	4	30
src/libraries/SignedFixedPointMathLib.sol	2	56
src/libraries/Structs.sol	7	563
src/ModuleRegistry.sol	2	39
src/modules/Fees.sol	30	259
src/modules/Funding.sol	12	100
src/modules/MarginRatios.sol	2	229
src/modules/PriceImpact.sol	11	71
src/modules/Pricing.sol	10	76
src/Perp.sol	43	628
src/PerpFactory.sol	25	179
src/PerpGuardHook.sol	32	113
src/ProtocolFeeManager.sol	3	46
src/WithdrawalController.sol	7	230

§ 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					
L-01	No in-place release path for excess insurance	Suggestion	LOW	ACKNOWLEDGED	p. 10
L-02	MakerMoved does not index `posId`	Events	LOW	RESOLVED	p. 11
L-03	Maker liquidation lifecycle events report zero liquidity delta	event-accounting	LOW	RESOLVED	p. 12
L-04	Phantom close reserve underpays terminal short liquidators	logic-error	LOW	RESOLVED	p. 13
L-05	Deferred flip fee debit under-haircuts second-hop fee claims	Business logic errors	LOW	RESOLVED	p. 14
L-06	Split maker liquidations can overpay rewards	Gaming	LOW	ACKNOWLEDGED	p. 16
L-07	Dynamic LP fee depends on swap partitioning	Gaming	LOW	ACKNOWLEDGED	p. 23
I-01	Unused `MoveMakerParams` import remains in `PerpLogic`	Superfluous Code	INFORMATIONAL	RESOLVED	p. 25
I-02	Negative settled cash blocks solvent zero-residual maker closes	DoS	INFORMATIONAL	RESOLVED	p. 26
I-03	MarginTransferred can be emitted without a USDC transfer	Events	INFORMATIONAL	RESOLVED	p. 27
I-04	Redundant tick-change check before event emission	Gas Optimization	INFORMATIONAL	RESOLVED	p. 28
I-05	Extreme reduce-only deltas revert with an arithmetic panic	Error	INFORMATIONAL	RESOLVED	p. 29
I-06	Widening the price-impact band can make later utilization accrual exceed transaction gas limits	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 30
I-07	Zero-liquidity maker adjustments perform an unnecessary Uniswap call	Gas Optimization	INFORMATIONAL	RESOLVED	p. 31
I-08	OpenInterestUpdated can be emitted without an open-interest change	Events	INFORMATIONAL	RESOLVED	p. 32
I-09	Market creation logs omit the initial governor	Events	INFORMATIONAL	RESOLVED	p. 33
I-10	Utilization can be increased via matched positions	Gaming	INFORMATIONAL	ACKNOWLEDGED	p. 34

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-11	`PerpMakerLogic.moveMaker()` allows positions below floor	Documentation	INFORMATIONAL	ACKNOWLEDGED	p. 35
I-12	Overlapping liquidation constraints create a temporary taker no-action window	Trust Assumptions	INFORMATIONAL	ACKNOWLEDGED	p. 36
I-13	`reduceOnly` documentation implies strict exposure reduction	Documentation	INFORMATIONAL	RESOLVED	p. 38
I-14	Missing taker position-state bounds let partial liquidation alter pending flips and backstop terms	Frontrunning	INFORMATIONAL	RESOLVED	p. 39
I-15	Partial liquidation leaves unfillable short dust	DoS	INFORMATIONAL	ACKNOWLEDGED	p. 41
I-16	Relaxed convergence paths can exceed the transaction gas limit	denial-of-service	INFORMATIONAL	ACKNOWLEDGED	p. 43
I-17	Principal rounding can make mark health overstate immediate closeability	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 45
I-18	Healthy maker liquidation attempts modify liquidity before reverting	Gas Optimization	INFORMATIONAL	RESOLVED	p. 46
I-19	Full maker exits use a redundant refresh flag	Best Practices	INFORMATIONAL	RESOLVED	p. 47
REMEDIATION REVIEW					
I-01	Taker flips settle swap fees per leg but utilization reserve per action	Unexpected Behavior	INFORMATIONAL	ACKNOWLEDGED	p. 48
I-02	Atomic taker flips retain equity outside the withdrawal-controller path	Documentation	INFORMATIONAL	ACKNOWLEDGED	p. 50
I-03	Funding correction is split by raw liquidity	Accounting	INFORMATIONAL	ACKNOWLEDGED	p. 52

LOW L-01 ACKNOWLEDGED

No in-place release path for excess insurance

CATEGORY	Suggestion
LOCATION	`src/libraries/PerpLogic.sol:1046`
REVIEW	Main Review

DESCRIPTION

The updated utilization accounting routes fees attributable to open interest above in-band maker closing capacity into a reserve. On taker settlement, `PerpLogic.realizeReserve` uses that amount to repay bad debt first and credits the remainder to `feeFund.insurance`. In the current implementation, `accountBadDebt` is the only path that debits the insurance bucket, and it does so only to cover a realized loss.

If a healthy market accumulates insurance materially above its desired loss buffer, the excess cannot be redistributed to traders or LPs through the current interface. It remains reserved, uses TVL-cap headroom, and can be consumed only by future bad debt and can therefore become some type of stucked tokens if the insurance funds grows bigger than reasonable. This is a conditional capital-efficiency limitation.

RECOMMENDATION

Be aware and document this behavior or consider to implement a way to release excess insurance funds if ever necessary.

RESOLUTION

Status: Acknowledged.

LOW **L-02** RESOLVED

MakerMoved does not index `posId`

CATEGORY	Events
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/interfaces/IPerp.sol#L130-L132
REVIEW	Main Review

DESCRIPTION

`MakerMoved` includes `posId`, but it is not marked `indexed` like other maker lifecycle events. An indexer cannot filter move logs for one position by topic and must fetch and decode all `MakerMoved` logs to find that position.

RECOMMENDATION

Be aware of and document this behavior or consider marking `posId` as `indexed` in `MakerMoved`.

RESOLUTION

Status: Resolved.

LOW L-03

RESOLVED

Maker liquidation lifecycle events report zero liquidity delta

CATEGORY	event-accounting
LOCATION	`packages/mirror-strobelabs-perpcity-contracts/src/libraries/PerpLiquidationLogic.sol:L119-L170`
REVIEW	Main Review

DESCRIPTION

`PerpLiquidationLogic.liquidateMaker()` emits `MakerAdjusted`, `MakerConverted` and `MakerClosed` with `liquidityDelta = 0` despite removing nonzero liquidity, while the voluntary path reports the actual signed change.

Consumers replaying lifecycle liquidity deltas can retain incorrect maker liquidity.

RECOMMENDATION

Emit the actual negative removed liquidity in each liquidation lifecycle event.

PROOF OF CONCEPT `https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/maker-liquidation-lifecycle-events-report-zero-liquidity-del-f20df39d.sol`

RESOLUTION

Status: Resolved.

LOW **L-04** RESOLVED

Phantom close reserve underpays terminal short liquidators

CATEGORY	logic-error
LOCATION	packages/mirror-strobelabs-perpcity-contracts/src/libraries/PerpLiquidationLogic.sol:liquidateTaker:L299-L309
REVIEW	Main Review

DESCRIPTION

A terminal short liquidation may leave an unfillable rounding remainder that is written off at mark before the position is deleted. The reward calculation nevertheless subtracts a hypothetical remainder-close reserve from the equity cap. No subsequent close is required, so this unnecessarily reduces the keeper's reward and increases the owner's payout

RECOMMENDATION

Compute the remainder close reserve only for nonterminal liquidations.

```
-    if (netDelta.amount0() != 0) {
+    if (!terminal && netDelta.amount0() != 0) {
         (uint256 sqrtMin, uint256 sqrtMax) = s.modules.priceImpact
             .sqrtPriceBounds(
                 snap.spots.ammPrice, snap.spots.index, snap.emas.ammPrice,
                   snap.emas.index, snap.markPrice
             );
         reserve = takerCloseReserve(
             netDelta.amount0(), valAfter, sqrtMin, sqrtMax, s.modules.fees.maxSwapFee()
               + MAX_PROTOCOL_FEE
         );
     }
```

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/phantom-close-reserve-underpays-terminal-short-liquidators-585fa33f.sol

RESOLUTION

Status: Resolved.

LOW L-05 RESOLVED

Deferred flip fee debit under-haircuts second-hop fee claims

CATEGORY	Business logic errors
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs/perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L286-L310 https://github.com/GuardianOrg/mirror-strobelabs/perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpAmmLogic.sol#L175-L195
REVIEW	Main Review

DESCRIPTION

`PerpLogic.adjustTaker()` executes a position flip by closing the existing side and opening the opposite side through two consecutive `PerpAmmLogic.swap()` calls.

```
SwapResult memory closeSr = PerpAmmLogic.swap(s, poolId, -
  int256(oldPerpBalance), snap, false);
closeQuote = closeSr.delta.amount1();
(snap.tick, snap.sqrtAmmPrice,) = PerpAmmLogic.poolState(poolId);
sr = PerpAmmLogic.swap(s, poolId, newPerpBalance, snap, false);
sr = _addSwapResults(closeSr, sr);
```

Each call immediately runs `PerpAmmLogic.accrueSwapFees()`. This socializes the protocol and creator fees against `badDebt` and credits the surviving protocol, creator, and insurance fees to `FeeFund`.

However, the first swap's `feeFundFeeAmt` remains included in `totalMargin` until both swaps are combined:

```
int256 netMargin = preSettledMargin + p.marginDelta - sr.totalFeeAmt;
removeSwapFeesFromMargin(s, sr);
```

When `badDebt` remains after the first swap, the second swap calculates its fee haircut using an overstated `totalMargin`. It therefore applies a smaller haircut and credits excess fees to `FeeFund`. The later aggregate debit corrects `totalMargin`, but it does not correct the excess fee claim or the remaining `badDebt`.

The PoC uses the deployed fee schedule and measures 9,733 USDC atoms of excess fee claims per flip in a large synthetic debt scenario. These claims leave the same amount of additional `badDebt` and reduce the backing available for later payouts.

RECOMMENDATION

Remove the first swap's `feeFundFeeAmt` from `totalMargin` before executing the second swap. After the second swap, remove only its newly created `feeFundFeeAmt`. Preserve the

L-05 Deferred flip fee debit under-haircuts second-hop fee claims CONTINUED

aggregate `SwapResult` fields used by the adjustment event and add a regression test against a per-hop accounting reference.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/deferred-flip-fee-debit-under-haircuts-the-second-swap-17177412.sol

RESOLUTION

Status: Resolved.

LOW **L-06** ACKNOWLEDGED

Split maker liquidations can overpay rewards

CATEGORY	Gaming
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLiquidationLogic.sol#L53-L68 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L541-L559
REVIEW	Main Review

DESCRIPTION

`PerpLiquidationLogic.liquidateMaker()` calculates the liquidation reward with `PerpLogic.makerMarkLiqFeeBase()`:

```
uint256 markFeeBase =
    PerpLogic.makerMarkLiqFeeBase(pos.delta, sqrtPl, sqrtPu, oldLiquidity, liquidityAmount,
      snap.markPrice);
```

`PerpLogic.makerMarkLiqFeeBase()` assumes that the liquidity is removed at the mark price:

```
(uint256 perps, uint256 usd) =
    LiquidityAmounts.getAmountsForLiquidity(sqrtMark, sqrtPriceL, sqrtPriceU,
      liquidityAmount);
Delta afterDelta = delta + toBalanceDelta(perps.toInt128(), usd.toInt128());
```

The actual removal happens at the AMM price and the returned tokens are added to `pos.delta`:

```
Delta delta = PerpAmmLogic.modifyLiquidity(posId, maker.tickLower, maker.tickUpper, -
  liquidityAmount.toInt128());

int256 marginBeforeLiqFee = pos.margin.toInt256() + lpFees.toInt256() - funding;
pos.delta = pos.delta + delta;
```

After a partial liquidation, the hypothetical mark-priced result is discarded. The next liquidation calculates its reward from the actual AMM-settled `pos.delta`. When the mark and AMM prices differ, splitting the same removal can therefore change the total reward.

Example:

L-06 Split maker liquidations can overpay rewards CONTINUED

1. The maker is liquidatable while the mark and AMM prices remain unchanged
2. Removing `2,612,000,000` liquidity in one call pays `13.590887 USDC`
3. The position is reset to the same starting snapshot
4. Removing `2,369,000,000` and then `243,000,000` liquidity pays `14.423248 USDC`
5. Both calls remain liquidatable and pass `healthImproved()`
6. Splitting the removal pays the liquidator an additional `0.832361 USDC`, or approximately `6.12%`

This comparison only applies when the external state remains unchanged. Liquidations after a mark-price change, AMM swap, funding accrual, or another position action may correctly produce different rewards.

The previous finding, "Split maker liquidations repeatedly charge unchanged residual notional," had a different cause. The old formula charged a proportional share of the whole position value for every chunk. This repeatedly included the same residual exposure.

The [previous fix](https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/commit/6e19b9b1f99ac7d3ab85e47f991dc675a5f177d7) changed the fee base to `valBefore - valAfter`. The previous chunk's `valAfter` became the next chunk's `valBefore`, so split fees added up to the one-shot fee.

The later [mark-pricing change](https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/commit/2b4e44b10c87bee93e4bac5360755bfd64059bfe) protected the fee from AMM price manipulation. However, the hypothetical `afterDelta` is not stored, so it does not become the next chunk's starting state. This reintroduced split overpayment through a different accounting path.

RECOMMENDATION

Store a per-position maker liquidation fee correction containing both token amounts, a fixed episode start timestamp, and a fixed episode start mark price.

For each partial maker liquidation:

1. Reset an existing correction if a certain time interval has elapsed since the episode started or the mark has deviated by more than a given threshold from the episode start mark.
2. Calculate the reward from `pos.delta + correction`.
3. Execute the removal and update both correction amounts by `markDelta - actualDelta`.
4. Set the episode timestamp and mark only when starting a new episode. Do not roll either value after later chunks.
5. Continue using the real `pos.delta` for settlement, health, funding, and PnL.
6. Clear the correction when the resulting maker is no longer liquidation-eligible, after

L-06 Split maker liquidations can overpay rewards CONTINUED

a repair restores health, after `PerpLiquidationLogic.backstopMaker()`, and when the maker closes or converts into a taker.
7. Preserve the correction across liquidity adjustments that merely improve an unhealthy maker without ending the liquidation episode. Apply their actual token delta equally to the real and fee-accounting states.

RECOMMENDED FIX

```
diff --git a/src/libraries/PerpLiquidationLogic.sol b/src/libraries/PerpLiquidationLogic.sol
index 9279cd4..0fdf054 100644
--- a/src/libraries/PerpLiquidationLogic.sol
+++ b/src/libraries/PerpLiquidationLogic.sol
@@ -16,7 +16,17 @@ import {PerpFundingLogic} from "./PerpFundingLogic.sol";
 import {PerpLogic} from "./PerpLogic.sol";
 import {PerpMakerLogic} from "./PerpMakerLogic.sol";
 import {SignedFixedPointMathLib} from "./SignedFixedPointMathLib.sol";
-import {Maker, MakerFunding, PerpStorage, Position, PricePair, Snapshot, SwapResult, Taker}
   from "./Structs.sol";
+import {
+    Maker,
+    MakerFunding,
+    MakerLiquidationCorrection,
+    PerpStorage,
+    Position,
+    PricePair,
+    Snapshot,
+    SwapResult,
+    Taker
+} from "./Structs.sol";

 /// @dev Liquidation and backstop logic, split into its own linked library so it does not
   contribute to
 /// PerpLogic's deployed code size. Function bodies are moved verbatim from PerpLogic and
   reuse PerpLogic's
@@ -54,8 +64,10 @@ library PerpLiquidationLogic {
             PerpLogic.valPnl(pos.delta, snap.sqrtAmmPrice, sqrtPl, sqrtPu, maker.liquidity,
               snap.markPrice);
         // Fee base is mark-clamped inventory for this removal size; live valPnl stays for
           equity and the health
         // denominator so a pool push cannot inflate the nominal fee.
-        uint256 markFeeBase =
-            PerpLogic.makerMarkLiqFeeBase(pos.delta, sqrtPl, sqrtPu, oldLiquidity,
-              liquidityAmount, snap.markPrice);
+        MakerLiquidationCorrection memory correction =
+          PerpLogic.makerLiquidationCorrection(s, posId, snap.markPrice);
+        uint256 markFeeBase = PerpLogic.makerMarkLiqFeeBase(
+            pos.delta + correction.delta, sqrtPl, sqrtPu, oldLiquidity, liquidityAmount,
+              snap.markPrice
+        );
         MakerFunding memory makerFunding =
             PerpLogic.makerCumlFunding(s, maker.tickLower, maker.tickUpper, snap.tick,
               sqrtPu);
         int256 funding = PerpLogic.makerFundingAccrued(s, pos, maker, makerFunding, sqrtPl,
           sqrtPu);
```

L-06 Split maker liquidations can overpay rewards CONTINUED

```
@@ -64,6 +76,11 @@ library PerpLiquidationLogic {

         Delta delta = PerpAmmLogic.modifyLiquidity(posId, maker.tickLower, maker.tickUpper,
           -liquidityAmount.toInt128());

+        uint160 sqrtMark = uint160(snap.markPrice.mulSqrt(Q96).clamp(sqrtPl, sqrtPu));
+        (uint256 markPerps, uint256 markUsd) = LiqAmts.getAmountsForLiquidity(sqrtMark,
           sqrtPl, sqrtPu, liquidityAmount);
+        correction.delta = correction.delta + toBalanceDelta(markPerps.toInt128(),
           markUsd.toInt128()) - delta;
+        s.makerLiquidationCorrections[posId] = correction;
+
         int256 marginBeforeLiqFee = pos.margin.toInt256() + lpFees.toInt256() - funding;
         pos.delta = pos.delta + delta;
         pos.lastCumlFundingX96 = s.cumls.fundingX96;
@@ -115,6 +132,7 @@ library PerpLiquidationLogic {
             require(healthImproved(equityBefore, valBefore, netMargin + pnlAfter,
               valAfter), Errors.HealthNotImproved());
             s.positions[posId] = pos;
             s.makers[posId] = maker;
+            PerpLogic.clearHealthyMakerLiquidationCorrection(s, posId, pos, maker, snap);
             PerpLogic.refreshRatesAndEmas(s, poolId, snap.spots, snap.emas);
             emit IPerp.MakerAdjusted(
                 posId,
@@ -170,7 +188,10 @@ library PerpLiquidationLogic {
             emit IPerp.MakerClosed(posId, owner, 0, 0, funding, lpFees, pos.delta, 0, 0, 0,
               MakerFunding(0, 0));
             delete s.positions[posId];
         }
-        if (maker.liquidity == 0) delete s.makers[posId];
+        if (maker.liquidity == 0) {
+            delete s.makers[posId];
+            delete s.makerLiquidationCorrections[posId];
+        }

         emit IPerp.MakerLiquidated(posId, liquidationFeeRecipient, liquidityAmount,
           liqFeeAmt.toUint256());
         // Outgoing margin transfer: existing insolvency socialization can still haircut
           the fee.
@@ -220,6 +241,7 @@ library PerpLiquidationLogic {

         PerpLogic.refreshRatesAndEmas(s, poolId, snap.spots, snap.emas);
         PerpLogic.transferMargin(s, marginIn.toInt128(), address(this));
+        delete s.makerLiquidationCorrections[posId];
         emit IPerp.MakerBackstopped(
             posId,
             marginIn,
diff --git a/src/libraries/PerpLogic.sol b/src/libraries/PerpLogic.sol
index c913b8e..ba62349 100644
--- a/src/libraries/PerpLogic.sol
+++ b/src/libraries/PerpLogic.sol
@@ -1,6 +1,7 @@
 // SPDX-License-Identifier: GPL-3.0-or-later
 pragma solidity 0.8.30;

+import {TickMath} from "@uniswap/v4-core/src/libraries/TickMath.sol";
```

L-06 Split maker liquidations can overpay rewards CONTINUED

```
 import {BalanceDelta as Delta, toBalanceDelta} from "@uniswap/v4-
   core/src/types/BalanceDelta.sol";
 import {PoolId} from "@uniswap/v4-core/src/types/PoolId.sol";
 import {LiquidityAmounts} from "@uniswap/v4-core/test/utils/LiquidityAmounts.sol";
@@ -36,6 +37,7 @@ import {
     FundingState,
     Maker,
     MakerFunding,
+    MakerLiquidationCorrection,
     MoveMakerParams,
     OpenInterest,
     OpenTakerParams,
@@ -534,6 +536,45 @@ library PerpLogic {
         return nominalReward < availableEquity ? nominalReward : availableEquity;
     }

+    /// @dev Episode anchors never roll forward with subsequent liquidation chunks.
+    function makerLiquidationCorrection(PerpStorage storage s, uint256 posId, uint256
      markP)
+        internal
+        view
+        returns (MakerLiquidationCorrection memory correction)
+    {
+        correction = s.makerLiquidationCorrections[posId];
+        if (
+            correction.markPrice == 0 || block.timestamp - correction.startedAt > 5 minutes
+                || (markP > correction.markPrice ? markP - correction.markPrice :
                  correction.markPrice - markP)
+                    > correction.markPrice.fullMulDiv(2, 100)
+        ) {
+            correction = MakerLiquidationCorrection(Delta.wrap(0), block.timestamp, markP);
+        }
+    }
+
+    /// @dev Only fee accounting uses the correction; health and settlement retain real
      inventory.
+    function clearHealthyMakerLiquidationCorrection(
+        PerpStorage storage s,
+        uint256 posId,
+        Position memory pos,
+        Maker memory maker,
+        Snapshot memory snap
+    ) internal {
+        uint160 sqrtPl = TickMath.getSqrtPriceAtTick(maker.tickLower);
+        uint160 sqrtPu = TickMath.getSqrtPriceAtTick(maker.tickUpper);
+        (uint256 val, int256 pnl) =
+            valPnl(pos.delta, snap.sqrtAmmPrice, sqrtPl, sqrtPu, maker.liquidity,
              snap.markPrice);
+        MakerLiquidationCorrection memory correction = makerLiquidationCorrection(s, posId,
          snap.markPrice);
+        uint256 fee = makerMarkLiqFeeBase(
+                pos.delta + correction.delta, sqrtPl, sqrtPu, maker.liquidity,
                  maker.liquidity, snap.markPrice
+            )
+            .fullMulDiv(s.modules.fees.liqFee(), E6);
+        (, uint24 liqMr,) = s.modules.marginRatios.makerMarginRatios();
```

L-06 Split maker liquidations can overpay rewards CONTINUED

```
+        if (isHealthy(pos.margin.toInt256() + pnl - fee.toInt256(), val, liqMr)) {
+            delete s.makerLiquidationCorrections[posId];
+        }
+    }
+
     /// @dev Nominal maker liquidation fee base for removing `liquidityAmount`: the maker
       value removed, excluding PnL
     /// lost solely because V4 floors each principal withdrawal. Inventory is taken at the
       mark clamped into the range,
     /// not at the live pool price. Settlement still uses real modifyLiquidity amounts;
       only the fee uses this
diff --git a/src/libraries/PerpMakerLogic.sol b/src/libraries/PerpMakerLogic.sol
index f6ddd5c..c6bd9a8 100644
--- a/src/libraries/PerpMakerLogic.sol
+++ b/src/libraries/PerpMakerLogic.sol
@@ -176,6 +176,7 @@ library PerpMakerLogic {

             s.positions[p.posId] = pos;
             s.makers[p.posId] = maker;
+            PerpLogic.clearHealthyMakerLiquidationCorrection(s, p.posId, pos, maker, snap);
             PerpLogic.refreshRatesAndEmas(s, poolId, snap.spots, snap.emas, snap.markPrice,
               p.liquidityDelta != 0);
             emit IPerp.MakerAdjusted(
                 p.posId,
@@ -250,6 +251,7 @@ library PerpMakerLogic {
             }

             delete s.makers[p.posId];
+            delete s.makerLiquidationCorrections[p.posId];
         }
     }

@@ -342,6 +344,7 @@ library PerpMakerLogic {

         s.positions[p.posId] = pos;
         s.makers[p.posId] = maker;
+        PerpLogic.clearHealthyMakerLiquidationCorrection(s, p.posId, pos, maker, snap);

         PerpLogic.refreshRatesAndEmas(s, poolId, snap.spots, snap.emas);
         emit IPerp.MakerMoved(
diff --git a/src/libraries/PerpPreviewLogic.sol b/src/libraries/PerpPreviewLogic.sol
index 3499371..c9139df 100644
--- a/src/libraries/PerpPreviewLogic.sol
+++ b/src/libraries/PerpPreviewLogic.sol
@@ -69,7 +69,12 @@ library PerpPreviewLogic {
             (fullLiqVal,) =
                 PerpLogic.valPnl(pos.delta + toBalanceDelta(perps.toInt128(),
                   usd.toInt128()), snap.markPrice);
             markFeeBase = PerpLogic.makerMarkLiqFeeBase(
-                pos.delta, sqrtPl, sqrtPu, maker.liquidity, maker.liquidity, snap.markPrice
+                pos.delta + PerpLogic.makerLiquidationCorrection(s, posId,
                   snap.markPrice).delta,
+                sqrtPl,
+                sqrtPu,
+                maker.liquidity,
+                maker.liquidity,
```

L-06 Split maker liquidations can overpay rewards CONTINUED

```
+                snap.markPrice
             );
         } else {
             (int256 total, int256 funding, uint256 util) =
               PerpLogic.takerFeesAccrued(cumls, s.takers[posId], pos);
diff --git a/src/libraries/Structs.sol b/src/libraries/Structs.sol
index a7b1606..d9deacf 100644
--- a/src/libraries/Structs.sol
+++ b/src/libraries/Structs.sol
@@ -126,12 +126,20 @@ struct PerpStorage {
     FundingState fundingState;
     /// @dev Strict bounds used to allocate the elapsed utilization interval.
     UtilizationBand utilizationBand;
+    mapping(uint256 => MakerLiquidationCorrection) makerLiquidationCorrections;
 }

 /*////////////////////////////////////////////////////////////////////////////
                                POSITION STATE
 ////////////////////////////////////////////////////////////////////////////*/

+/// @dev Fee-only inventory adjustment for a fixed maker liquidation episode.
+struct MakerLiquidationCorrection {
+    BalanceDelta delta;
+    uint256 startedAt;
+    uint256 markPrice;
+}
+
 /// @dev Maker-specific state for an active liquidity range.
 struct Maker {
     int24 tickLower;
```

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/split-maker-liquidations-overcharge-the-liquidation-reward-869b5a97.sol

RESOLUTION

Status: Acknowledged.

LOW **L-07** ACKNOWLEDGED

Dynamic LP fee depends on swap partitioning

CATEGORY	Gaming
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpAmmLogic.sol#L288-L300
REVIEW	Main Review

DESCRIPTION

`_accrueSwapSegmentFees()` obtains the LP fee from `fees()` using the pre-swap AMM spot price and applies that rate to all volume executed by the external swap. The LP fee grows with the squared deviation between the AMM spot price and its EMA.

When a taker trades toward the EMA, they can split one intended swap between `openTaker()` and subsequent `adjustTaker()` calls. Each later call begins closer to the EMA and is charged a lower LP fee, whereas a single swap over the same price path applies the higher initial rate to all volume. The resulting position exposure and AMM price are the same, apart from expected rounding.

The included PoC uses the deployment-script fee curve and 2% price-impact bound. Splitting a swap into ten calls reduces LP fees by `717` USDC atoms, while improving the taker's net margin by `702` USDC atoms after per-call rounding. The effect requires a pre-existing spot/EMA divergence and is small under the intended configuration, but it makes LP fee revenue depend on execution partitioning rather than only the traded price path.

RECOMMENDATION

If partition-independent dynamic fees are desired, calculate the dynamic LP fee over each executed price interval rather than applying an entry-price quote to all volume. Extend the fee-module interface to return the fee amount over a constant-liquidity `sqrtPrice` interval using the shared EMA reference. The price-dependent component should be integrated over the executed path, such that concatenating adjacent paths yields the same fee as executing their union in one swap, subject only to rounding.

Alternatively, if allowing traders to select the lower-cost execution mode when moving toward the EMA is acceptable, acknowledge this behavior and document it as an intended fee-model property in `TRUST_ASSUMPTIONS.md`.

PROOF OF CONCEPT https://github.com/GuardianOrg/mirror-GuardianOrg-mirror-strobelabs-perpcity-contracts-1cac74c3f2a1eb19d711e47b0d68638f/blob/main/test/unit/perp/DynamicFeePartition.t.sol

L-07 Dynamic LP fee depends on swap partitioning CONTINUED

RESOLUTION

Status: Acknowledged.

INFORMATIONAL **I-01** RESOLVED

Unused `MoveMakerParams` import remains in `PerpLogic`

CATEGORY	Superfluous Code
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L32-L40
REVIEW	Main Review

DESCRIPTION

`PerpLogic` imports `MoveMakerParams`, but never references the type. The move-maker entry point and parameter handling are implemented in `Perp.sol` and `PerpMakerLogic.sol`, so this import is stale.

RECOMMENDATION

Be aware and consider to remove unused code.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-02 RESOLVED

Negative settled cash blocks solvent zero-residual maker closes

CATEGORY	DoS
LOCATION	`src/libraries/PerpMakerLogic.sol:130`
REVIEW	Main Review

DESCRIPTION

When a maker owes accrued funding, `adjustMaker` deducts it from cash collateral: `preSettledMargin = pos.margin + lpFees - funding` (line 127). With `marginDelta = 0`, this signed cash balance can be negative. A full liquidity removal can leave zero residual perps (`amount0 == 0`) and a positive quote balance (`amount1`) large enough to cover the cash deficit and leave positive final equity. The quote balance is the offsetting value; funding is the debit.

However, line 130 casts `preSettledMargin + marginDelta` to `uint256` before adding the removal delta to `pos.delta` at line 131 and before the zero-perp close branch at lines 230-249 adds `amount1` to cash and pays out. Negative cash therefore reverts `Overflow()` even if final equity is positive. A maker that remains above the liquidation and backstop thresholds cannot use either path to exit and must add USDC to bring cash nonnegative before closing.

RECOMMENDATION

Be aware and consider to document this needed top up or to rewrite this logic to allow closes in that case and avoid the overflow.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-03 RESOLVED

MarginTransferred can be emitted without a USDC transfer

CATEGORY	Events
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L1055-L1068
REVIEW	Main Review

DESCRIPTION

`transferMargin` emits `MarginTransferred` even when `marginDelta == 0`. A successful `adjustMaker` or `moveMaker` call with zero margin delta reaches this path, so the event can appear without moving USDC or changing `totalMargin`. Log consumers should not treat the event alone as proof that funds moved.

RECOMMENDATION

Be aware of and document the zero-transfer case.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-04 RESOLVED

Redundant tick-change check before event emission

CATEGORY	Gas Optimization
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpAmmLogic.sol#L400
REVIEW	Main Review

DESCRIPTION

`_crossTicksAndAccrueSwapFees()` returns early whenever `snap.tick == endingTick`. Therefore, execution reaching the `TicksCrossed` event is guaranteed to satisfy `snap.tick != endingTick`, making the additional condition redundant and adding unnecessary bytecode and runtime gas.

```
if (snap.tick != endingTick) emit IPerp.TicksCrossed(snap.tick, endingTick, zeroForOne);
```

RECOMMENDATION

Remove the redundant condition and emit `TicksCrossed` unconditionally after the traversal loop.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-05 RESOLVED

Extreme reduce-only deltas revert with an arithmetic panic

CATEGORY	Error
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/Perp.sol#L707
REVIEW	Main Review

DESCRIPTION

`adjustTaker` calls `_checkReduceOnly` when `reduceOnly` is true. `_checkReduceOnly` adds the caller-supplied `int256 delta` to the existing `int128` position balance before checking whether the request reduces exposure. For an existing long, `delta = type(int256).max` overflows that addition; for an existing short, `delta = type(int256).min` underflows it. Both revert with Solidity `Panic(0x11)` before `Errors.ReduceOnlyViolated()`.

These are invalid requests and each revert is atomic. The effect is an inconsistent revert selector for integrations that recognize a reduce-only violation by its custom error; no valid reduction or market state was shown to be affected.

RECOMMENDATION

Be aware and consider to check whether `delta` lies in the reduction interval for the current position before adding it and reject with a custom error instead to follow best practices.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-06 ACKNOWLEDGED

Widening the price-impact band can make later utilization accrual exceed transaction gas limits

CATEGORY	Informational
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpUtilizationLogic.sol#L81-L85
REVIEW	Main Review

DESCRIPTION

`setPriceImpactModule` accrues utilization under the existing cached price-impact band before installing the replacement module and caching its new band. After time passes, utilization rewards are distributed across initialized ticks in that cached band. Widening can therefore make a later accrual much more expensive than the module update itself.

For example:
- Taker open interest exists, and makers add many narrow ranges outside the current band but inside a proposed wider band.
- The governor installs the wider module. The update accrues against the old narrow band, then caches the wider band and completes.
- After time passes, `touch()` or a position action must distribute rewards across the newly included ticks. If the band admits enough initialized ticks, that accrual can exceed the transaction gas limit.
- Replacing the wide module with a narrow one also accrues against the cached wide band first, so an ordinary rollback can fail for the same reason.

Part of this is known in the trust assumptions therefore this risk is only pointed out as info.

RECOMMENDATION

Be aware and consider enforcing a tighter maximum band width.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-07 RESOLVED

Zero-liquidity maker adjustments perform an unnecessary Uniswap call

CATEGORY Gas Optimization

LOCATION https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpMakerLogic.sol#L117

REVIEW Main Review

DESCRIPTION

`adjustMaker()` calls `PerpAmmLogic.modifyLiquidity()` even when `p.liquidityDelta == 0`. This performs an unnecessary Uniswap V4 unlock, hook authorization, position update, and settlement cycle. PerpCity independently settles its custom LP fees, while native Uniswap fee growth and `feeDelta` are expected to remain zero, so the zero-liquidity call has no useful effect.

RECOMMENDATION

Skip `PerpAmmLogic.modifyLiquidity()` when `p.liquidityDelta == 0` and use a zero `Delta` while continuing the protocol's funding, fee, margin, checkpoint, and health accounting.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-08 RESOLVED

OpenInterestUpdated can be emitted without an open-interest change

CATEGORY	Events
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L731-L737
REVIEW	Main Review

DESCRIPTION

`adjustTaker` accepts `perpDelta == 0` for a margin-only adjustment. The zero-swap path leaves the taker's perp balance unchanged, yet `_applyTakerSwapState` still rewrites the same open interest and emits `OpenInterestUpdated`. Log consumers should not treat this event alone as proof that open interest changed.

RECOMMENDATION

Be aware of and document the unchanged-open-interest case.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-09 RESOLVED

Market creation logs omit the initial governor

CATEGORY	Events
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/interfaces/IPerpFactory.sol#L27-L38
REVIEW	Main Review

DESCRIPTION

`createPerp` takes an initial `governor`, and `Perp.initialize` stores it, but neither initialization nor `PerpCreated` emits that address. A log-only monitor cannot identify a market's initial governor without reading creation calldata or historical state.

RECOMMENDATION

Consider including the initial governor in `PerpCreated` or emitting `SetGovernor` during initialization.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-10 ACKNOWLEDGED

Utilization can be increased via matched positions

CATEGORY	Gaming
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L449-L456
REVIEW	Main Review

DESCRIPTION

`PerpLogic.refreshRatesAndEmas()` derives long and short utilization independently from gross `oi.long` and `oi.short`. A participant can open approximately matched opposing positions and increase both utilization rates.

The second swap can restore the AMM price, utilization capacity, and the stored `physicalCap` close to their original values through `rotatePhysicalCapacity()`. The matched positions nevertheless remain recorded as gross open interest. Consequently, subsequent calls to `PerpLogic.accrue()` charge higher utilization fees to all takers, while `PerpUtilizationLogic.accrue()` and `split()` distribute the eligible portion to makers. A participant controlling a sufficiently large share of eligible maker liquidity can therefore recover much of the fees paid by their matched taker positions and receive a share of the additional fees paid by unrelated takers.

RECOMMENDATION

Document that utilization is intentionally based on gross, independently maintained long and short OI, including the resulting behavior for matched opposing positions. Monitor matched gross OI and eligible maker concentration.

PROOF OF CONCEPT https://github.com/GuardianOrg/mirror-GuardianOrg-mirror-strobelabs-perpcity-contracts-1cac74c3f2a1eb19d711e47b0d68638f/blob/poc/economic-capture-915821a/test/unit/perp/EconomicCapture.t.sol

RESOLUTION

Status: Acknowledged.

INFORMATIONAL **I-11** ACKNOWLEDGED

`PerpMakerLogic.moveMaker()` allows positions below floor

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpMakerLogic.sol#L286
REVIEW	Main Review

DESCRIPTION

`PerpMakerLogic.moveMaker()` permits a maker's resulting settled margin to remain below `MIN_OPENING_MARGIN` whenever `marginDelta >= 0`. The resulting position must still satisfy the maker initial margin ratio because the function calls `PerpLogic.checkActionHealth()` with `repairEligible` set to `false`. This matches the conditional floor applied by `PerpMakerLogic.adjustMaker()` and `PerpLogic.adjustTaker()`. However, section 4.13 of `TRUST_ASSUMPTIONS.md` explicitly documents this behavior only for `PerpMakerLogic.adjustMaker()` and `PerpLogic.adjustTaker()`, omitting `PerpMakerLogic.moveMaker()`.

RECOMMENDATION

Update section 4.13 of `TRUST_ASSUMPTIONS.md` to include `PerpMakerLogic.moveMaker()` and clarify that existing positions must retain `MIN_OPENING_MARGIN` only when withdrawing margin, while risk-increasing actions remain governed by the applicable initial margin ratio.

Alternatively, modify the check in `PerpMakerLogic.moveMaker()` so it always requires the floor to be satisfied.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-12

ACKNOWLEDGED

Overlapping liquidation constraints create a temporary taker no-action window

CATEGORY Trust Assumptions

LOCATION https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLiquidationLogic.sol#L312-L315 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L329-L338 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpAmmLogic.sol#L124-L127

REVIEW Main Review

DESCRIPTION

`TRUST_ASSUMPTIONS.md` separately accepts that partial taker liquidations can revert with `NegativeMargin`, full liquidations can revert with `PriceImpactTooHigh`, and backstopping starts only below the backstop ratio. These constraints can overlap and temporarily leave an unhealthy position with no executable liquidation or repair path.

Funding and utilization can make `settledMargin` negative while favorable PnL keeps full equity positive. `PerpLiquidationLogic.liquidateTaker()` checks the cash-only `netMargin` before checking whether a partial liquidation improves health:

```
require(netMargin >= 0, Errors.NegativeMargin());
```

`PerpLogic.adjustTaker()` applies the same check before `PerpLogic.checkActionHealth()`. A holder reduction that improves full-equity health still reverts unless the holder first clears the entire cash deficit.

A full liquidation does not use this cash floor, but it must close through `PerpAmmLogic.swap()`. It reverts when the close moves the AMM outside the mark-relative price-impact band. `PerpLiquidationLogic.backstopTaker()` is unavailable while the position remains above the lower backstop ratio.

Example:

1. Funding and utilization reduce settled cash to approximately `-0.72 USDC`
2. Favorable PnL leaves approximately `+2.92 USDC` full equity on `40 USDC` notional
3. The position is below the liquidation ratio but above the backstop ratio
4. Band-fillable partial liquidations and no-withdrawal holder reductions revert with `NegativeMargin`
5. Full liquidation reverts with `PriceImpactTooHigh`
6. Backstopping reverts with `NotLiquidatable`

I-12 Overlapping liquidation constraints create a temporary taker no-action window CONTINUED

RECOMMENDATION

`TRUST_ASSUMPTIONS.md:134-163` describes the no-action window in the finding, but it doesn't mention the negative cash as a reason, only the health check. Acknowledge the finding if this behavior is acceptable.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/clarify-the-negative-cash-floor-for-taker-partial-liquidatio-270d9cfd.sol

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-13 RESOLVED

`reduceOnly` documentation implies strict exposure reduction

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/Perp.sol#L705-L717
REVIEW	Main Review

DESCRIPTION

`Perp._checkReduceOnly()` permits unchanged exposure when `delta == 0`, allowing margin-only adjustments. However, its documentation states that it checks whether the delta reduces the position, which may lead integrators to expect a strict exposure reduction.

RECOMMENDATION

Document that `reduceOnly` prevents increasing or flipping exposure but permits unchanged exposure and margin-only adjustments.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-14

RESOLVED

Missing taker position-state bounds let partial liquidation alter pending flips and backstop terms

CATEGORY	Frontrunning
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/Structs.sol#L414-L421 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/Perp.sol#L423-L430 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/Perp.sol#L448-L455 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLiquidationLogic.sol#L375-L392
REVIEW	Main Review

DESCRIPTION

`Perp.adjustTaker()` applies `AdjustTakerParams.perpDelta` to the position's live exposure. `amt1Limit`, `maxTotalFee`, and `deadline` limit execution price, fees, and time, but do not limit the starting or resulting exposure.

A previous remediation added `reduceOnly`. This protects reductions and closes because `Perp._checkReduceOnly()` reverts if the transaction crosses zero. It does not protect an intentional flip because every flip must set `reduceOnly` to `false`.

A permissionless `PerpLiquidationLogic.liquidateTaker()` call can therefore reduce the position before a pending flip executes. The unchanged `perpDelta` closes the smaller remaining position and opens a larger opposite position than the holder intended.

Example:

1. A holder has a `5,000,000` atom long and submits `perpDelta = -5,000,001` to open a one-atom short
2. Without a liquidation, the position ends at `-1`
3. A liquidator first closes `1,000,000` atoms
4. The same pending `perpDelta` now leaves a `-1,000,001` atom short
5. After the index moves from $46 to $50, the intended one-atom short remains healthy and realizes `8.455040` USDC
6. The enlarged short becomes liquidatable and returns only `2.957831` USDC to the holder

`Perp.backstopTaker()` has the same missing state commitment. A backstopper commits to a fixed `marginIn`, but cannot require minimum remaining exposure, minimum acquired equity, or an expected position state. A partial taker liquidation can consume equity and reduce exposure before the pending backstop executes. If the original `marginIn` still satisfies the live health check, the backstopper pays the same amount for the changed position.

I-14 Missing taker position-state bounds let partial liquidation alter pending flips and backstop terms

CONTINUED

RECOMMENDATION

Add an expected current exposure or position-state nonce to `AdjustTakerParams` and reject the transaction if the position changed before execution. Keep `reduceOnly` for reductions and closes.

Acknowledge the `Perp.backstopTaker()` case the same way the [backstopMaker](https://app.notion.com/p/guardianaudits/M-18-3ad8bda5828c802984f2f235a6059da3?source=copy_link) one was.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/missing-position-state-bounds-let-liquidation-change-pending-38595b02.sol

RESOLUTION

Status: Resolved.

INFORMATIONAL I-15 ACKNOWLEDGED

Partial liquidation leaves unfillable short dust

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/mirror-GuardianOrg-mirror-strobelabs-perpcity-contracts-1cac74c3f2a1eb19d711e47b0d68638f/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLiquidationLogic.sol#L279-L328 https://github.com/GuardianOrg/mirror-GuardianOrg-mirror-strobelabs-perpcity-contracts-1cac74c3f2a1eb19d711e47b0d68638f/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpAmmLogic.sol#L102-L138
REVIEW	Main Review

DESCRIPTION

It is an intended behavior that a successful partial liquidation may reduce exposure enough to make the residual position healthy, after which further liquidation attempts revert with `NotLiquidatable`.

The `TRUST_ASSUMPTIONS.md` also record that V4 rounding can make reported short-closing capacity slightly exceed executable sell-side liquidity. Voluntary closes remain all-or-nothing, residual open interest can reserve maker capacity, and operations may need to supply replacement liquidity.

These behaviors can be combined through the caller-selected `perpAmount` in `PerpLiquidationLogic.liquidateTaker()`. Consider an unhealthy short for which a full liquidation would succeed: `PerpAmmLogic.swap()` would buy the available perp, verify that the unfilled amount does not exceed the capacity ledger's pre-existing overstatement, settle that remainder at the mark, and close the position.

Instead, a permissionless liquidator can request a near-full partial liquidation equal to the amount the AMM can execute. The partial trade fills exactly and consumes the remaining sell-side liquidity, but leaves a rounding-sized short. Because most exposure was removed while margin remains, the residue can become healthy.

The two accepted constraints then form a no-action state:

- Further liquidation reverts with `NotLiquidatable` because the residue is healthy.
- The owner's close through `Perp.adjustTaker()` reverts with `InsufficientLiquidityToFill` because voluntary closes require an exact fill.
- The residual short open interest can prevent the final maker from removing all liquidity.

The distinct behavior is therefore not either constraint individually, but that a permissionless partial liquidation can transform a position that was terminally closable into the already accepted liquidity-rescue state.

I-15 Partial liquidation leaves unfillable short dust CONTINUED

RECOMMENDATION

Be aware of the combinations of these two behaviors and the discrepancy they cause between a full and partial liquidation.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/partial-liquidation-leaves-unfillable-short-dust-requiring-r-9e44e535.sol

RESOLUTION

Status: Acknowledged.

INFORMATIONAL **I-16** ACKNOWLEDGED

Relaxed convergence paths can exceed the transaction gas limit

CATEGORY	denial-of-service
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpAmmLogic.sol#L86-L130
REVIEW	Main Review

DESCRIPTION

The protocol recognizes that swap gas grows with the number of initialized ticks crossed, but assumes that the configured price-impact band and fixed tick spacing bound this cost. Fixed tick spacing only bounds tick density within a given price interval. The interval itself can become substantially larger when the AMM is outside the ordinary price-impact band.

`PerpAmmLogic.swap()` obtains the normal `sqrtMin` and `sqrtMax` bounds and then relaxes whichever bound excludes the pre-swap `snap.sqrtAmmPrice`. If the AMM is below the band, the effective interval becomes `[snap.sqrtAmmPrice, sqrtMax]`; if it is above the band, it becomes `[sqrtMin, snap.sqrtAmmPrice]`. A convergence-direction swap can therefore traverse the entire displacement between the stranded AMM and the band, in addition to part or all of the ordinary band.

This exposure is shared by every sufficiently large convergence-direction call to `PerpAmmLogic.swap()`. It therefore applies to taker opens through `PerpLogic.openTaker()`, increases, reductions, closes, and flips through `PerpLogic.adjustTaker()`, and liquidations through `PerpLiquidationLogic.liquidateTaker()`. These paths execute the V4 swap and replay the crossed ticks for protocol accounting. A terminal short liquidation additionally scans the path through `PerpAmmLogic._topOfBook()`, making liquidation the most expensive and consequential affected flow.

Permissionless makers can populate the expanded convergence interval with minimum-margin narrow ranges and make taker operations exceed the maximum transaction gas budget.

RECOMMENDATION

If this is accepted behavior, update `TRUST_ASSUMPTIONS.md` to document that convergence relaxation can make `PerpAmmLogic.swap()` traverse every initialized tick between an out-of-band AMM price and the relaxed band edge.

I-16 Relaxed convergence paths can exceed the transaction gas limit CONTINUED

Operators can monitor both this price gap and the number of initialized candidate ticks along the corrective path, using benchmarked thresholds for a gas-safe traversal.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/fragmented-convergence-paths-can-exceed-the-liquidation-tran-8960cb2e.sol

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-17

ACKNOWLEDGED

Principal rounding can make mark health overstate immediate closeability

CATEGORY	Informational
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLogic.sol#L238-L265
REVIEW	Main Review

DESCRIPTION

V4 rounds base input up and quote output down for every segment crossed by an exact-input long close. `PerpLogic.openTaker()` checks initial health using the mark value from `PerpLogic.valPnl()` without accounting for this closing cost.

In the PoC, a healthy 36-atom long opened across 36 ranges at a $500,000 mark costs 17,164,454 quote atoms. Immediately closing it returns only 8,813,863 quote atoms. The loss exceeds its 5,396,075 settled margin and the close reverts with `NegativeEquity`. The same position closes successfully against continuous liquidity.

The reverted transaction leaves the position open. The owner must add collateral or wait for liquidity or prices to change.

RECOMMENDATION

Document in `TRUST_ASSUMPTIONS.md` that mark health is not an executable close quote and that fragmented V4 rounding can add approximately one base atom of closing loss per crossed range. Position previews and integrations should estimate the full close and warn when its expected cost exceeds the available collateral.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/refinery/poc-01a15467-3593-72f8-9a16-b312b0ea8388/test/refinery-poc/fragmented-principal-rounding-admits-healthy-longs-whose-imm-22a963ea.sol

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-18 RESOLVED

Healthy maker liquidation attempts modify liquidity before reverting

CATEGORY Gas Optimization

LOCATION https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpLiquidationLogic.sol#L65-L104

REVIEW Main Review

DESCRIPTION

`PerpLiquidationLogic.liquidateMaker()` calls `PerpAmmLogic.modifyLiquidity()` and updates accounting before checking eligibility through `PerpLogic.isHealthy()`. Since all eligibility inputs are already available, healthy attempts perform unnecessary PoolManager and accounting work before reverting.

RECOMMENDATION

Move the eligibility check before `PerpAmmLogic.modifyLiquidity()`, aligning the flow with `PerpLiquidationLogic.liquidateTaker()`.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-19 RESOLVED

Full maker exits use a redundant refresh flag

CATEGORY	Best Practices
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/libraries/PerpMakerLogic.sol#L193-L235 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/915821ace0cd08841c207a5055aba7a59064cd3c/src/Perp.sol#L366-L371
REVIEW	Main Review

DESCRIPTION

The zero-liquidity branch in `PerpMakerLogic.adjustMaker()` calls the six-parameter `refreshRatesAndEmas()` overload with `p.liquidityDelta != 0` as `poolMutated`. Because `Perp.adjustMaker()` requires existing maker liquidity to be nonzero, reaching this branch means `p.liquidityDelta` necessarily removed all liquidity and modified the pool. The expression is therefore always `true` and obscures this invariant.

RECOMMENDATION

Continue using the six-parameter `refreshRatesAndEmas()` overload, but pass `true` directly for `poolMutated`. This preserves `snap.markPrice`, avoids an additional pricing-module call, and makes the full-removal invariant explicit.

RESOLUTION

Status: Resolved.

INFORMATIONAL **I-01** ACKNOWLEDGED

Taker flips settle swap fees per leg but utilization reserve per action

CATEGORY	Unexpected Behavior
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/6e830949c5e3c0fc417cd1f9a2ab56ace4df8a8e/src/libraries/PerpLogic.sol#L136-L162
REVIEW	Remediation Review

DESCRIPTION

When `PerpLogic.adjustTaker()` flips a taker position from long to short or vice versa, it executes a closing swap followed by an opening swap. The function finalizes each leg's fee-fund claims separately by calling `PerpLogic.removeSwapFeesFromMargin()` after each `PerpAmmLogic.swap()`:

```
        if (oldPerpBalance != 0 && newPerpBalance != 0 && (oldPerpBalance < 0) !=
          (newPerpBalance < 0)) {
            // Close first to realize the old side's exact quote, then open the new side
              from the post-close pool.
            // Both legs quote rates from the pre-trade prices but round fees independently.
              The second leg's
            // price-impact anchor advances to the intermediate pool price.
            SwapResult memory closeSr = PerpAmmLogic.swap(s, poolId, -
              int256(oldPerpBalance), snap, false);
            closeQuote = closeSr.delta.amount1();
            // Finalize the close leg's fee claims so the open leg's haircut does not count
              them as active collateral.
            removeSwapFeesFromMargin(s, closeSr);
            (snap.tick, snap.sqrtAmmPrice,) = PerpAmmLogic.poolState(poolId);
            sr = PerpAmmLogic.swap(s, poolId, newPerpBalance, snap, false);
            removeSwapFeesFromMargin(s, sr);
            sr = _addSwapResults(closeSr, sr);
```

However, the old position's pending utilization reserve is realized only after both swap legs have completed:

```
        (int256 accruedFees, int256 funding, uint256 util) = takerFeesAccrued(s, taker,
          pos);
        uint256 reserveCredit = realizeReserve(s, p.posId, taker, pos.delta.amount0());
        int256 preSettledMargin = pos.margin.toInt256() - accruedFees;
        int256 netMargin = preSettledMargin + p.marginDelta - sr.totalFeeAmt;
        removeReserveFromMargin(s, reserveCredit);
```

I-01 Taker flips settle swap fees per leg but utilization reserve per action CONTINUED

Consequently, the operation uses two different settlement boundaries: swap fees are finalized per leg, while utilization reserve is finalized once per complete adjustment. Insurance credit and bad-debt repayment attribution can consequently depend on whether the taker flips atomically or uses two actions.

RECOMMENDATION

Document that the current ordering prioritizes preserving/rebuilding insurance by imposing losses on protocol and creator fee claims, even when pending reserve would have been sufficient to cover the debt if realized first.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-02

ACKNOWLEDGED

Atomic taker flips retain equity outside the withdrawal-controller path

CATEGORY	Documentation
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/6e830949c5e3c0fc417cd1f9a2ab56ace4df8a8e/src/libraries/PerpLogic.sol#L136-L147 https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/6e830949c5e3c0fc417cd1f9a2ab56ace4df8a8e/src/libraries/PerpLogic.sol#L219-L221
REVIEW	Remediation Review

DESCRIPTION

`PerpLogic.adjustTaker()` implements a side flip by executing a closing swap followed immediately by an opening swap while retaining the existing position and its margin:

```
        if (oldPerpBalance != 0 && newPerpBalance != 0 && (oldPerpBalance < 0) !=
          (newPerpBalance < 0)) {
            // Close first to realize the old side's exact quote, then open the new side
              from the post-close pool.
            // Both legs quote rates from the pre-trade prices but round fees independently.
              The second leg's
            // price-impact anchor advances to the intermediate pool price.
            SwapResult memory closeSr = PerpAmmLogic.swap(s, poolId, -
              int256(oldPerpBalance), snap, false);
            closeQuote = closeSr.delta.amount1();
            // Finalize the close leg's fee claims so the open leg's haircut does not count
              them as active collateral.
            removeSwapFeesFromMargin(s, closeSr);
            (snap.tick, snap.sqrtAmmPrice,) = PerpAmmLogic.poolState(poolId);
            sr = PerpAmmLogic.swap(s, poolId, newPerpBalance, snap, false);
            removeSwapFeesFromMargin(s, sr);
            sr = _addSwapResults(closeSr, sr);
```

In contrast, performing the same economic transition as a complete close followed by a new open first routes the closed position's remaining equity through `PerpLogic.closeTaker()`:

```
    /// @dev Pays remaining equity and deletes the position and taker checkpoints.
    function closeTaker(PerpStorage storage s, address equityRecipient, uint256 posId,
      uint256 equity) internal {
        transferMargin(s, -equity.toInt128(), equityRecipient);
        delete s.positions[posId];
        delete s.takers[posId];
    }
```

The outgoing `PerpLogic.transferMargin()` call applies any outstanding bad-debt haircut and routes the resulting USDC payout through the withdrawal controller. Depending on controller capacity and configuration, the payout may be transferred immediately or escrowed as a

I-02 Atomic taker flips retain equity outside the withdrawal-controller path CONTINUED

delayed claim. The user may therefore need additional external capital before opening the replacement position.

An atomic flip does not create an external payout, so it neither consumes withdrawal-controller capacity nor creates an escrowed claim. It also retains the position's nominal equity internally instead of realizing the bad-debt haircut that a complete close would apply.

Additionally, the new documentation for [GA-29](https://ops.guardian.security/engagements/01a13c39-4b18-7cda-b964-ec7fc1a66915/findings?round=01a13c39-4b3c-7df6-b2a5-6aaa421d042e&finding=01a183ab-5570-7a29-b2d2-313cdf9a8333) advises users to first close their position and then reopen it if they want to avoid an in-between liquidation resulting in an increased exposure. However, they may need to provide additional funds if the rate limiter cannot satisfy their closing request.

RECOMMENDATION

Document that an atomic taker flip retains collateral within the market and does not invoke the withdrawal controller, while a standalone close routes equity through the controller before a subsequent open and keep in mind the `GA-29` behavior. Frontends and integrations comparing the two paths should account for possible payout escrow, withdrawal capacity, and bad-debt haircuts. No change to the flip implementation is required.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-03 ACKNOWLEDGED

Funding correction is split by raw liquidity

CATEGORY	Accounting
LOCATION	https://github.com/GuardianOrg/mirror-strobelabs-perpcity-contracts/blob/6e830949c5e3c0fc417cd1f9a2ab56ace4df8a8e/src/libraries/PerpFundingLogic.sol#L108-L109
REVIEW	Remediation Review

DESCRIPTION

`PerpFundingLogic.accrue()` charges funding on the rounding gap between raw perp balances and maker inventory, and shares the offsetting correction across all makers by raw liquidity:

```
        int256 totalCorrectionX128 = (-exposureX96).sFullMulDiv(fundingAccruedX96, 1 << 64,
          true);
        cumls.fundingCorrectionX128 += totalCorrectionX128.sFullMulDiv(1,
          state.makerLiquidity, true);
```

A range entirely below the current price holds no perp inventory but still counts in `state.makerLiquidity`. Near `MIN_TICK` such liquidity is almost free, so a maker posting `MIN_OPENING_MARGIN` can hold nearly all of it and collect nearly all of the correction while funding is negative. The sign of the correction follows the sign of funding, which is observable on-chain: the maker can open the range while the correction is a credit and close it before funding turns positive.

The maker receives the share that would otherwise go to the other makers. The maker is credited while `emas.ammPrice` is below `emas.index`. With nearly the whole correction captured, the amount taken is:

```
|gap| × |fundingPerDay| × days open
fundingPerDay = clamp((emas.ammPrice − emas.index) × fundingRateMultiplier, ± markPrice ×
  maxFundingPerDayPercent)
```

`gap` is `baseX96 + inventoryX96` in perp atoms. It grows by up to about one atom per liquidity change or initialized tick crossed, and never clears. It can also be grown at another maker's expense. Each partial `PerpLiquidationLogic.liquidateMaker()` call floors the perps returned, so a caller who slices a liquidatable maker into removals holding under one perp atom moves that amount from the maker into the gap on every call.

The spread and the cap are both price amounts, so for the same percentage spread a higher-priced market yields a proportionally larger amount. A 150-atom gap at a price of 5,000 with funding at 2% per day for 30 days yields about 0.45 USDC.

I-03 Funding correction is split by raw liquidity CONTINUED

The spread has no hard limit, but funding is capped at `maxFundingPerDayPercent` of the mark, 5% in the deployment script. If arbitrage keeps the pool at the mark, the spread settles within `maxPremiumRatio`, 2% in the deployment script. Traders collecting funding should narrow it further, which shrinks the credit without turning it into a charge. That happens only if the spread turns positive, and the maker can observe the EMAs and close the range first.

The linked PoC runs the production configuration at a price of 100,000 with the pool at the mark and a 0.5% spread. Over 30 days the out-of-range range collects 2.26 USDC, which is exactly what the in-range makers stop receiving.

A [second PoC](https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/00fee218958e56bfc760af5d8c601bd1373d488b/test/refinery-poc/funding-correction-worst-case-at-max-index-3f70755a.sol) shows a worst case, on a market where the price is `MAX_INDEX_X96`, valuing one perp atom at ~1 USDC. There, 1,000 liquidity resizes leave a gap of about 1,000 USDC notional, more than the market's own inventory. With funding held at the premium clamp for 440 days, the maker collects 8,005 USDC. The charged maker cannot be liquidated because its residual exceeds the available close depth, so the loss reaches other makers' collateral.

RECOMMENDATION

`TRUST_ASSUMPTIONS.md` already states that the funding correction is socialized across makers. If this is accepted, consider adding that the share follows raw liquidity, so a range holding no perp inventory can collect most of it, and that the amount grows with the price per perp atom.

PROOF OF CONCEPT https://github.com/GuardianOrg/SentryRepo-PerpCity-Contracts-Updates--Round-1-01a153b6ed2573bc9aeb4fb8d6132288/blob/00fee218958e56bfc760af5d8c601bd1373d488b/test/refinery-poc/funding-correction-captured-by-out-of-range-liquidity.sol

RESOLUTION

Status: Acknowledged.

§ 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, or read more at **guardian.security**.

This report and its findings are also published at guardian.security/portfolio/perp-city/perpcity-contracts-updates-2026-11-27.