Pendle Core v3 Security Review

Auditors

Desmond Ho, Lead Security Researcher

Saw-mon and Natalie, Lead Security Researcher

RustyRabbit, Security Researcher

Ethan, Associate Security Researcher

Report prepared by: Lucas Goiriz

August 22, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Pendle is a permissionless yield-trading protocol where users can execute various yield-management strategies.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Pendle Core v3 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 53 days in total, Pendle Finance engaged with Spearbit to review the pendle-core-v3. The review was structured in the following way:

Review Structure

Length (days)	Scope	Commit Hashes	Team
22	Pendle V3 (Scope 1: Market + Markethub)	`d479e3...2681`	Desmond Ho, Saw-mon and Natalie, RustyRabbit, Ethan
10	Pendle V3 (Scope 2: AMM + Router)	`5a1249...54fb`	Desmond Ho, Saw-mon and Natalie, RustyRabbit
10	Pendle V3 Fix Review 1	`4f7694...5a26`	Desmond Ho, Saw-mon and Natalie, RustyRabbit
2	Pendle V3 Fix Review 2	`8f9855...b7d3`, `e2aa73...aa6c`	Desmond Ho, Saw-mon and Natalie, RustyRabbit
9	Pendle V3 ***[PRIVATE REVIEW]***	`41ca3b...8e11`	Saw-mon and Natalie

In this period of time a total of **53** issues were found in the following categories:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	5	3	2
Low Risk	17	6	11
Gas Optimizations	7	0	7
Informational	24	9	15
Total	**53**	**18**	**35**

5 Findings

6 Pendle Core v3

6.1 Medium Risk

6.1.1 Treasury withdrawals will revert for non-18 token decimals

Severity: Medium Risk

Context: Storage.sol#L147-L151

Description: `treasuryCash` is in scaled amount, but it's not unscaled to the raw amounts for withdrawals. Hence, it'll revert for tokens with `scalingFactor` > 1, i.e. tokens that have less than 18 decimals, like USDC.

Proof of Concept: Git apply the patch below.

```
  diff --git a/test/core/markethub/fees.t.sol b/test/core/markethub/fees.t.sol
  index 5d17ccb..b2c285f 100644
-  -- a/test/core/markethub/fees.t.sol
+  ++ b/test/core/markethub/fees.t.sol
  @@ -4,14 +4,18 @@ pragma solidity ^0.8.28;
   import {IRouter} from "boros/interfaces/IRouter.sol";
   import {PMath} from "boros/lib/math/PMath.sol";
   import {LONG, SHORT} from "boros/types/Order.sol";
-  import {TokenId} from "boros/types/MarketTypes.sol";
+  import {TokenId, MarketId} from "boros/types/MarketTypes.sol";
   import {TradeLib} from "boros/types/Trade.sol";
-
+  import {CustomDecimalsToken} from "src/test-helpers/MockToken.sol";
+  import {IMarketOff} from "boros/interfaces/IMarket.sol";
   import {BorosTestBase} from "test/BorosTestBase.sol";
   import {RouterWrapper} from "src/test-helpers/RouterWrapper.sol";
+  import {MarketSettingWrapper} from "src/test-helpers/MarketSettingWrapper.sol";
+  import {IERC20Errors} from "@openzeppelin/contracts/interfaces/draft-IERC6093.sol";

   contract MarketHubTest_Fees is BorosTestBase {
       using RouterWrapper for IRouter;
+      using MarketSettingWrapper for IMarketOff;
       using PMath for uint256;
       using PMath for int256;

  @@ -115,4 +119,58 @@ contract MarketHubTest_Fees is BorosTestBase {
           assertEq(marketHub.treasuryCash(tokenId), 0);
           assertEq(collateralAsset.balanceOf(treasury), treasuryCash);
       }
+
+      function test_exoticTokenWithdrawTreasury() public {
+          CustomDecimalsToken mockUsdc = new CustomDecimalsToken("USDC", "USDC", 6);
+          vm.startPrank(admin);
+          marketHub.registerToken(address(mockUsdc));
+          TokenId mockUsdcId = marketHub.tokenToId(address(mockUsdc));
+          indexOracle = deployFIndexOracle(maturity, getNextMarketAddress(), period,
↪  maxFIndexUpdateDelay);
+          address market1Address = marketFactory.create(
+              "PendleBoros Market 1",
+              "BOROS-1",
+              false,
+              maturity,
+              mockUsdcId,
+              tickStep,
+              genMarketConfig(address(indexOracle), address(rateOracle)),
```

```
+              defaultMarketImpliedRateInit()
+          );
+          registerMarket(market1Address);
+          IMarketOff market1 = IMarketOff(market1Address);
+          (, , MarketId market1Id, , ) = market1.descriptor();
+          market1.setMarginConfig(IMFactor, MMFactor);
+          market1.setFeeRates(takerFeeRate, otcFeeRate);
+          market1.setLiquidationIncentive(liquidationIncentiveFactor, 0);
+          vm.stopPrank();
+          changeFIndexOracleSettleFee(indexOracle, settleFeeRate);
+
+          uint256 rawAmount = 10_000e6;
+
+          vm.startPrank(alice);
+          router.enterMarket(true, market1Id);
+          mockUsdc.allocateTo(alice, rawAmount);
+          mockUsdc.approve(address(router), rawAmount);
+          router.vaultTransfer(mockUsdcId, rawAmount, true);
+
+          router.placeOrderALO(true, market1Id, LONG, 1 ether, 0);
+
+          vm.startPrank(bob);
+          router.enterMarket(true, market1Id);
+          mockUsdc.allocateTo(bob, rawAmount);
+          mockUsdc.approve(address(router), rawAmount);
+          router.vaultTransfer(mockUsdcId, rawAmount, true);
+
+          router.placeOrderFOK(true, market1Id, SHORT, 1 ether, type(int16).min);
+
+          uint256 treasuryCash = marketHub.treasuryCash(mockUsdcId);
+          assertGt(treasuryCash, 0);
+
+          TokenId[] memory tokenIds = new TokenId[](1);
+          tokenIds[0] = mockUsdcId;
+
+          vm.startPrank(admin);
+          vm.expectPartialRevert(IERC20Errors.ERC20InsufficientBalance.selector);
+          marketHub.withdrawTreasury(tokenIds);
+      }
   }
```

Recommendation: Unscale the `amount` to the raw amount.

```
- IERC20(data.token).safeTransfer(TREASURY, amount);
+ IERC20(data.token).safeTransfer(TREASURY, amount / data.scalingFactor);
```

Pendle Finance: Fixed in commit 711a6d1b.

Spearbit: Fix verified.

6.1.2 Fractional division causes system to debit less than expected

Severity: Medium Risk

Context: PaymentLib.sol#L22, PaymentLib.sol#L28

Description: A key system invariant is that the sum of the `accCash` of all users and the `treasuryCash` for a given `tokenId` should be no more than the actual token balance held by the `MarketHub`. It is possible for this invariant to be broken due to under-debiting when a order is filled through multiple partial matches. There are 2 instances of this.

- The first, when calculating the `payment` when sweeping and processing settled orders, where `PMath.mulDown()` rounds to 0 for negative numbers.
- The second, when calculating the upfront fixed cost, which rounds towards 0 too.

Simple example for illustration: `22 / 7 = 3`, but `(-16 / 7) + (-6 / 7) = -2 + 0 = -2`.

Proof of Concept:

```
function test_multipleTradersCashSettlement() public {
    address charlie = makeUserWithCashAndEnterMarket0("charlie", 10 ether);
    // total accCash in system = 30 ether

    vm.startPrank(admin);
    // set fees to 0
    market0.setFeeRates(0, 0);
    // set mark rate to 10%
    setMarkRate(rateOracle, 10e16);
    vm.stopPrank();

    vm.prank(alice);
    router.placeOrderALO(true, market0Id, LONG, 11 ether, 1906); // ~10%
    vm.prank(bob);
    router.placeOrderFOK(true, market0Id, SHORT, 3.66 ether, type(int16).min);
    vm.prank(charlie);
    router.placeOrderFOK(true, market0Id, SHORT, 7.34 ether, type(int16).min);
    router.settlePaymentAndOrders(toCrossAcc(alice));
    router.settlePaymentAndOrders(toCrossAcc(bob));
    router.settlePaymentAndOrders(toCrossAcc(charlie));

    // try forwarding 1 epoch at a time
    FIndex index = indexOracle.getLatestFIndex();
    uint32 timestamp = index.fTime();
    while (timestamp < maturity) {
        int112 floatingRate = int112(8e16) * int112(uint112(period)) / int112(365 days);
        forwardOneEpoch(floatingRate);
        index = indexOracle.getLatestFIndex();
        timestamp = index.fTime();
    }

    vm.prank(alice);
    router.exitMarket(true, market0Id);

    vm.prank(bob);
    router.exitMarket(true, market0Id);

    vm.prank(charlie);
    router.exitMarket(true, market0Id);

    int256 aliceCashAfter = marketHub.accCash(toCrossAcc(alice));
    int256 bobCashAfter = marketHub.accCash(toCrossAcc(bob));
    int256 charlieCashAfter = marketHub.accCash(toCrossAcc(charlie));
    int256 treasuryCash = int256(marketHub.treasuryCash(tokenId));
    assertLe(
        aliceCashAfter +
        bobCashAfter +
        charlieCashAfter +
        treasuryCash,
        30 ether
    );
}
```

The test fails because the assertion breaks the invariant: the total amount accounted for is 1 wei more than

expected. Upon inspection of the logs, we find that the payments made is as such:

- Bob's payment is `-3660000000000000000 * 39525114155250813 / ONE = -144661917808217975`.
- Charlie's payment is `-7340000000000000000 * 39525114155250813 / ONE = -290114337899540967`.

The sum of Bob and Charlie's debits (`-144661917808217975 + (-290114337899540967) = -434776255707758942`) is 1 less than Alice's credit (`434776255707758943`). Likewise, for the upfront fixed cost.

```
function test_upfrontFixedCostRounding() public {
  address charlie = makeUserWithCashAndEnterMarket0("charlie", 10 ether);
  // total accCash in system = 30 ether

  // set all fees to 0
  vm.startPrank(admin);
  market0.setFeeRates(0, 0);
  vm.stopPrank();
  changeFIndexOracleSettleFee(indexOracle, 0);

  vm.prank(alice);
  router.placeOrderALO(true, market0Id, SHORT, 1 ether, 1906); // ~10%
  vm.prank(bob);
  router.placeOrderFOK(true, market0Id, LONG, 0.2745 ether, 1906);
  vm.prank(charlie);
  router.placeOrderFOK(true, market0Id, LONG, 0.7255 ether, 1906);
  router.settlePaymentAndOrders(toCrossAcc(alice));

  int256 aliceTotalValue = getTotalValue(alice);
  int256 bobTotalValue = getTotalValue(bob);
  int256 charlieTotalValue = getTotalValue(charlie);
  assertLe(aliceTotalValue + bobTotalValue + charlieTotalValue, 30 ether);
}
```

Recommendation: In general, the rounding directions need to be further scrutinised. A simple fix would be to have `mulDown()` round "up" for negative numbers, i.e. round to negative infinity instead of 0, but it is unclear the effect this will have for other `mulDown()` calls. Another approach would be to explicitly round to negative infinity for negative payments, so that more is taken.

```
// TODO: implement mulUp(): round to positive / negative infinity for positive / negative numbers
↪  respectively
function calcSettlement(int256 signedSize, FIndex last, FIndex current) internal pure returns (PayFee
↪  res) {
   if (last == current) return PayFeeLib.ZERO;
   int256 floatingDiff = int256(int64(current.floatingIndex() - last.floatingIndex()));
   int256 netPaymentSign = (signedSize >> 255) | int256(1) * (floatingDiff >> 255) | int256(1);
   res = PayFeeLib.from(
      netPaymentSign > 0 ?
      signedSize.mulDown(floatingDiff) :
      signedSize.mulUp(floatingDiff),
      signedSize.abs().mulDown(current.feeIndex() - last.feeIndex())
);
```

Similarly, for `calcUpfrontFixedCost()`.

Pendle Finance: We introduce new functions for multiplication and division, with different rounding direction, and use them as suggested.

Spearbit: Acknowledged. Reviewed in a subsequent audit.

6.1.3 First market exit conditional check is inadequate

Severity: Medium Risk

Context: MarginManager.sol#L48

Description: The first condition of requiring `margin.isZero()` and `signedSize == 0` isn't robust enough, as:

- It is unlikely but possible for both the market `markRate` and `minMarginIndexRate` to be `0`.
- It doesn't account the user's limit orders.

Thus, should both the `markRate` and `minMarginIndexRate` be 0, it allows a user to do the following:

- Use an account A to...
  1. Place a LONG / SHORT limit order (if interest rates are expected to be positive / negative respectively).
  2. Exit the market and withdraw all funds.
- Use an account B to fill that limit order and wait till maturity.

Since the user has already withdrawn all funds from account A and account B has a positive net trade, the earnings from that trade, when withdrawn, will come from other users' deposits.

Proof of Concept:

```
function test_placeOrderAndExitMarket() public {
  vm.startPrank(admin);
  market0.setMinMarginIndexRate(0); // set min margin index rate to 0
  setMarkRate(rateOracle, 0); // set mark rate to 0

  // Alice does the following:
  // 1. place a LONG limit order
  // 2. exit the market
  // 3. withdraw all cash
  vm.startPrank(alice);
  router.placeOrderALO(true, market0Id, LONG, 1 ether, 500);
  // Alice is able to exit market, withdraw all funds
  router.exitMarket(true, market0Id);
  router.vaultTransfer(tokenId, uint256(marketHub.accCash(toCrossAcc(alice))), false);
  assertEq(marketHub.accCash(toCrossAcc(alice)), 0);

  // Bob (Alice's alt account) fulfills the LONG order
  // because Bob is short, he's paid upfront fixed cost
  // hence basis of comparison is before order fulfillment
  int256 bobCashBefore = marketHub.accCash(toCrossAcc(bob));
  vm.startPrank(bob);
  router.placeOrderFOK(true, market0Id, SHORT, 1 ether, 500);
  vm.stopPrank();

  // fast forward to maturity
  vm.warp(maturity + 1 days);
  vm.prank(fIndexOracleUpdater);
  FIndex index = indexOracle.getLatestFIndex();
  uint32 timestamp = index.fTime();
  int112 floatingRate = int112(1e16) * int112(uint112(maturity - timestamp)) / int112(365 days);
  vm.prank(fIndexOracleUpdater);
  indexOracle.updateFloatingRate(floatingRate, maturity);

  vm.prank(bob);
  router.exitMarket(true, market0Id);
  // Bob is net positive
  int256 bobCashAfter = marketHub.accCash(toCrossAcc(bob));
  assertGt(bobCashAfter, bobCashBefore);
  assertEq(marketHub.accCash(toCrossAcc(alice)), 0);
}
```

Recommendation:

- Consider requiring `minMarginIndexDuration` to be non-zero.
- Consider returning and checking against `_getMaxPossibleAbsoluteSize()` instead of `signedSize`, so that limit orders are taken into account.

Pendle Finance: Fixed in commit 4f769483. We now explicitly check for `signedSize == 0` and `nOrders = 0`.

Spearbit: Fix verified.

6.2 Low Risk

6.2.1 `mulDown` and `divDown` for `int256` rounds towards `0`

Severity: Low Risk

Context: PMath.sol#L48-L58, PMath.sol#L71-L81, PMath.sol#L56, PMath.sol#L79

Description: `mulDown` and `divDown` for `int256` rounds towards `0` when `sign(x) * sign(y) == -1`, this due the fact that for the final calculation both of these functions use `sdiv` which for negative values it rounds towards `0` and not towards $-\infty$ as the `Down` keyword suggests.

Currently these functions with negative signatures are used in:

`mulDown(int256, int256)`:

location	safety comment
`_applyFee`	as long as `_storage.feeRate` is non-negative
`_calcLiqTrade`	needs to be investigated, since `sizeToLiquidator` could be negative
`_binSearchRemainingSize`	either one of `addWithBook.toSignedSize(side)` or `bookRate` could be negative
`_calculateTickCost`	`size.toSignedSize(side)` is negative for shorts
`_placeOrderOnBookAndAMMs`	???
`calcSettlement`	???
`calcPositionValue`	`signedSize` could be negative
`fromSizeAndRate`	`_signedSize` could be negative
`from3`	`_rate` could be negative ??

`divDown(int256, int256)`:

location	safety comment
`calcSwapOutput`	`normFixedIn` could be negative
`_settleProcessGetHealth`	???
`getUserInfo`	???

Recommendation: In all the above cases the desired rounding direction needs to be carefully analyzed.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.2.2 `tryRemove` may throw the wrong error when a cancellation reverts

Severity: Low Risk

Context: Tick.sol#L84-L95

Summary: Users may receive the wrong error if their cancellation reverts due to the order of the guard clauses in `Tick.tryRemove`.

Finding Description: There are three reasons that a user's attempt to cancel an order may revert: the order has already been filled, the order has already been cancelled, or the order has an invalid ID and cannot be found. To handle each error case, the function `tryRemove` first checks if the `orderIndex` is less than the `headIndex`, and if it is, it determines that the order has already been filled. Later, it checks the position's size to determine whether it has already been cancelled:

```
if (orderIndex < headIndex) {
    require(!isStrictCancel, Err.MarketOrderFilled());
    return (0, tickSum);
} else if (orderIndex >= tailIndex) {
    revert Err.MarketOrderNotFound();
}

uint128 removedSize = $.node[orderIndex].orderSize();
if (removedSize == 0) {
    require(!isStrictCancel, Err.MarketOrderCancelled());
    return (0, tickSum);
}
```

The problem is that every `orderIndex` - be it filled or cancelled - will eventually be less than the `headIndex`, since the `headIndex` continues to increase with each new filled order. In the event that the `headIndex` has increased beyond an already-cancelled `orderIndex`, the first case will evaluate as true, which will incorrectly throw with `Err.MarketOrderFilled`.

Impact Explanation: Users may receive the wrong error message when their attempts to cancel orders fail, leading them to take improper subsequent steps that could waste money and time.

Likelihood Explanation: The likelihood depends on the volume of a given market. This is guaranteed to happen eventually for any market with sufficient remaining time before maturity, but since users would likely not expect a cancellation to work for long after the order was initially placed, the `headIndex` would need to increase past their order's index fairly rapidly in order for this to cause real problems for users.

Proof of Concept: Add this test to `orders.t.sol`:

```
function test_cancelAlreadyCancelledOrderWithHeadIndexPastOrderId() public {
    // 1. place 2 orders
    vm.startPrank(alice);
    router.placeOrderALO(true, market0Id, LONG, 0.5 ether, 500);
    router.placeOrderALO(true, market0Id, LONG, 1 ether, 500);
    vm.stopPrank();

    // 2. cancel the order with smaller size
    OrderId[] memory toRemoveIds = new OrderId[](1);
    IMarketOff.Order[] memory orders = market0.getAllOpenOrders(toCrossAcc(alice));
    assertEq(orders.length, 2);
    assertEq(orders[0].size, 1 ether);
    assertEq(orders[1].size, 0.5 ether);
    toRemoveIds[0] = orders[1].id;
    LongShort memory cancels = LongShortLib.createCancels(toRemoveIds, false, true);
    vm.prank(address(router));
    marketHub.orderAndOtc(market0Id, toCrossAcc(alice), cancels, new OTCTrade[](0));

    // 3. fill the order of larger size
    vm.prank(bob);
```

```
    router.placeOrderFOK(true, market0Id, SHORT, 1 ether, 500);

    // 4. try cancelling the smaller order again, should revert with MarketOrderCancelled
    vm.prank(address(router));
    vm.expectRevert(Err.MarketOrderCancelled.selector, address(market0));
    marketHub.orderAndOtc(market0Id, toCrossAcc(alice), cancels, new OTCTrade[](0));
}
```

Recommendation: To preserve all three error states with minimal additional gas consumption, rearrange the order and implementation of the guard clauses:

```
require(orderIndex < tailIndex, Err.MarketOrderNotFound());

uint128 removedSize = $.node[orderIndex].orderSize();
if (isStrictCancel) {
   require(removedSize > 0, Err.MarketOrderCancelled());
   require(headIndex <= orderIndex, Err.MarketOrderFilled());
} else if (removedSize == 0 || orderIndex < headIndex) {
   return (0, tickSum);
}
```

Alternatively, if the additional gas cost is not deemed to be worth maintaining all three error conditions, this function could return a single error for both cases (e.g., `Err.MarketOrderFilledOrCancelled`). Since cancellations are expected to work properly in most cases, it is reasonable to pass the responsibility of checking the size of the order in question on to the affected user if they want to take further action.

Pendle Finance: Fixed in commit 62018338.

Spearbit: Fix verified.

6.2.3 PRNG seed is constant on Arbitrum

Severity: Low Risk

Context: SweepProcessUtils.sol#L60

Description: According to Arbitrum's docs, `block.prevrandao` returns a constant `1`. This is verified in the proof of concept below. This means that the seed is a constant, so the pivot position sequence doesn't change.

Proof of Concept:

```
contract CheckPrevrandao {
    error PrevRandaoIsOne();
    constructor() {
        uint256 prevrandao = block.prevrandao;
        if (prevrandao == 1) {
            revert PrevRandaoIsOne();
        }
    }
}
```

Running `forge create src/CheckPrevrandao.sol:CheckPrevrandao --private-key <PRIVATE_KEY> --rpc-url <ARBITRUM_RPC_URL>` will result in the following response:

```
Error: server returned an error response: error code 3: execution reverted, data: "0x084d9915"
```

which corresponds to the custom error `PrevRandaoIsOne()`.

Recommendation: Given the use-case, pseudo-randomness is sufficient. Hence, consider using `blockhash` of a previous block number.

Pendle Finance: Fixed in 4f769483. by changing the seed to `block.number`.

Spearbit: Fix verified.

6.2.4 `_disableInitializers()` is missing in `MiscModule`'s and `MarketFactory`'s constructor

Severity: Low Risk

Context: MiscModule.sol#L18-L32

Description: Based on the implementation of `MiscModule`, one can guess that this contract would leave behind an upgradable contract and that is why there is a `initialize` function defined as well. Moreover in the test repo's `setup` function one has:

```
miscModule = address(new MiscModule(address(permissionController), address(marketHub)));
// ...
address routerImplementation = address(new Router(ammModule, authModule, miscModule, tradeModule));
router = IRouter(
    address(
        new TransparentUpgradeableProxy(
            routerImplementation,
            admin,
            abi.encodeCall(IMiscModule.initialize, ("PendleBoros Router", "1.0", numTicksToTryAtOnce))
        )
    )
);
```

and so to conform to the patterns used in other implementation contracts in the codebase it would be best to call `_disableInitializers()` in the constructor.

Recommendation: Modify the `constructor` to:

```
constructor(
    address permissionController_,
    address marketHub_
) PendleRolesPlugin(permissionController_) TradeStorage(marketHub_) {
    _disableInitializers(); // <--- added
}
```

The same issue and recommendation applies to `MarketFactory`

Pendle Finance: Fixed in commit 711a6d1b.

Spearbit: Fix verified.

6.2.5 Events missing

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description/Recommendation:

- `MiscModule.sol#L31`: Emitting `NumTicksToTryAtOnceUpdated` is missing in the `MiscModule.initialize`.
- BaseAMM.sol#L84: Emitting `ImpliedRateObservationWindowUpdated` is missing.
- BaseAMM.sol#L83: Emitting `FeeRateUpdated` is missing.
- BaseAMM.sol#L77-L81: Emitting `AMMConfigUpdated` is missing.

Pendle Finance: Fixed in commit 70fc6ac6.

Spearbit: Fix verified.

6.2.6 A malicious `amm` can rewrite another AMM's market account

Severity: Low Risk

Context: MiscModule.sol#L98-L103

Description: In `setAMMIdToAcc` there is no check to make sure the `amm` address provider by the master admin comes from a group of verified AMM addresses. And thus if accidentally a rogue `amm` address is provided to this endpoint, this `amm` can return any values when its `AMM_ID()` and `SELF_ACC()` are called to potentially override values belonging to other AMM addresses.

Recommendation: It would be best to add a check to make sure the `amm` address provided belongs to a set of verified addresses. For example, one can check that this address was created using the `AMMFactory`'s `create` endpoint:

- Computing addresses using nonces of `AMMFactory` or...
- `AMMFactory` can keep a registry of its created `AMM` which can be queried by other contracts.

Footnote: It would also be best in this function to check whether the market connected to the `AMM` is one of the markets registered in market hub (or deployed by the market factory). This check is also missing when creating an `AMM` through the `AMMFactory.create`.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.2.7 `adjustedMinAbsRate` might be greater than `adjustedMaxAbsRate`

Severity: Low Risk

Context: PositiveAMMMath.sol#L166-L167

Description: In `calcSwapSize` one tweaks the rate ranges to:

$$r'_{min} = r_{min} \otimes (10^{18} + 10^{8})$$
$$r'_{max} = r_{max} \otimes (10^{18} - 10^{8})$$

Whenever r_{min} and r_{max} are set either in the `constructor` or through `setAMMConfig` the following checks are performed:

$$0 < r_{min} < r_{max}$$

And so it is possible that for a pair of (r_{min}, r_{max}) we would have.

$$r'_{min} > r'_{max}$$

If this happens in `calcSwapSize`, `targetRate` r_t under these circumstances would always be r'_{max}. In even more extreme cases this value $r_t = r'_{max}$ can be less than r_{min} the original `state.minAbsRate` value:

$$r_t = r'_{max} < r_{min} \leq r'_{min}$$

This could for example happen when ($\blacksquare r = r_{max} - r_{min}$):

$$(10^{10} - 1)\blacksquare r < r_{min}$$

or

$$\left(1 - \frac{1}{10^{10}}\right) r_{max} < r_{min} < r_{max}$$

Recommendation: The check in `_setAMMConfig` should be changed to:

```
require(
   0 < minAbsRate &&
   minAbsRate.tweakUp(1e8) < maxAbsRate.tweakDown(1e8), // <--- changed to a stricter inequality
Err.AMMInvalidRateRange());
```

Pendle Finance: Fixed in commit 8f9855c2.

Spearbit: Fix verified.

6.3 Gas Optimization

6.3.1 Redundant `onlyAllowed` modifier in `cashTransferAll()`

Severity: Gas Optimization

Context: MarginManager.sol#L73

Description: The `onlyAllowed` modifier for `cashTransferAll()` is redundant because it will be checked when it calls `cashTransfer()`.

Proof of Concept:

```
function test_cashTransferAll() public {
   MarketAcc mainCross = toCrossAcc(alice);
   MarketAcc mainIsolated = toIsolatedAcc(alice, market0Id);
   vm.prank(address(router));
   marketHub.cashTransferAll(mainCross, mainIsolated);
}
```

About 50 gas savings is observed.

Recommendation:

```
- function cashTransferAll(MarketAcc from, MarketAcc to) public onlyAllowed returns (int256 amountOut) {
+ function cashTransferAll(MarketAcc from, MarketAcc to) public returns (int256 amountOut) {
```

Pendle Finance: Aware & intentional, gas is not much of a concern here & we want to make it clear at a glimpse that every function is locked.

Spearbit: Acknowledged.

6.3.2 Cache `users[userID]` to a local variable

Severity: Gas Optimization

Context: MarketOrderAndOtc.sol#L165-L180

Description: `users[userID]` will be read multiple times throughout the function; caching it to a local variable will help save some gas.

Proof of Concept: Gas savings are as follows:

```
test_liquidateOneMarketWhileAnotherMatured() (gas: -79 (-0.000%))
test_liquidateOneMarketWhileAnotherMatured() (gas: -79 (-0.000%))
test_maturedMarket_zeroMarginAndPositionValue() (gas: -79 (-0.000%))
test_canRemoveLiquidityDualAfterMaturity() (gas: -79 (-0.000%))
test_canRemoveLiquidityDualAfterMaturity() (gas: -79 (-0.000%))
test_canRemoveLiquiditySingleAfterMaturity() (gas: -79 (-0.000%))
```

```
test_canRemoveLiquiditySingleAfterMaturity() (gas: -79 (-0.000%))
test_minMarginIndexDuration() (gas: -79 (-0.000%))
test_withBookAndOneAMM_numTicksToTryAtOnce_short() (gas: -1659 (-0.001%))
test_withBookAndOneAMM_numTicksToTryAtOnce_long() (gas: -1659 (-0.001%))
test_minMarginIndexRate() (gas: -79 (-0.002%))
test_liquidateMultipleMarkets() (gas: -79 (-0.002%))
test_liquidateMultipleMarkets() (gas: -79 (-0.002%))
test_canNotLiquidate() (gas: -79 (-0.003%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -158 (-0.004%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -158 (-0.004%))
test_liquidatorWeakIMCheck() (gas: -158 (-0.005%))
test_positionValueCalculation() (gas: -79 (-0.006%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -237 (-0.006%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -237 (-0.006%))
test_weakIMCheck() (gas: -158 (-0.007%))
test_withBookAndOneAMM_shortMultipleTicks() (gas: -79 (-0.008%))
test_withBookAndOneAMM_longMultipleTicks() (gas: -79 (-0.008%))
test_forceDeleverage() (gas: -79 (-0.010%))
test_weakIMCheck() (gas: -316 (-0.010%))
test_cancelOrderCheckIMWeak() (gas: -158 (-0.010%))
test_withBookAndOneAMM_short() (gas: -79 (-0.011%))
test_withBookAndOneAMM_long() (gas: -79 (-0.011%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -158 (-0.012%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -158 (-0.012%))
test_withBookAndOneAMM_notMatchLongLimitOrder() (gas: -79 (-0.012%))
test_withBookAndOneAMM_notMatchShortLimitOrder() (gas: -79 (-0.012%))
test_withBookAndOneAMM_fees() (gas: -79 (-0.012%))
test_withBookAndOneAMM_justEnoughWholeTick() (gas: -79 (-0.013%))
test_withBookAndOneAMM_takerFeeDiscount() (gas: -158 (-0.013%))
test_withBookAndOneAMM_otcFeeDiscount() (gas: -158 (-0.013%))
test_addLiquiditySingleWhenNeutral() (gas: -79 (-0.014%))
test_addLiquiditySingleWhenNeutral() (gas: -79 (-0.014%))
test_exitMarkets() (gas: -127 (-0.014%))
test_liquidate_returnValues() (gas: -79 (-0.014%))
test_liquidate_returnValues() (gas: -158 (-0.015%))
test_addLiquidityDualWhenNeutral() (gas: -79 (-0.015%))
test_addLiquidityDualWhenNeutral() (gas: -79 (-0.015%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -237 (-0.015%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -237 (-0.015%))
test_removeLiquiditySingle() (gas: -316 (-0.015%))
test_removeLiquiditySingle() (gas: -316 (-0.015%))
test_withBookAndOneAMM_gtcPartiallyFilled_notRevert() (gas: -79 (-0.015%))
test_orderAndOtc_returnValues() (gas: -256 (-0.017%))
test_addLiquiditySingle_justEnoughWholeTick() (gas: -158 (-0.017%))
test_addLiquiditySingle_justEnoughWholeTick() (gas: -158 (-0.017%))
test_slippage_addLiquiditySingle() (gas: -316 (-0.019%))
test_canAddLiquiditySingleWhenBelowIM() (gas: -158 (-0.020%))
test_slippage_addLiquiditySingle() (gas: -316 (-0.020%))
test_canAddLiquiditySingleWhenBelowIM() (gas: -158 (-0.020%))
test_basicPayment() (gas: -79 (-0.020%))
test_basicPayment() (gas: -79 (-0.020%))
test_maxInitialMarginViaOTC() (gas: -158 (-0.022%))
test_changeMarginConfig() (gas: -79 (-0.022%))
test_canRemoveLiquidityDualWhenBelowIM() (gas: -79 (-0.023%))
test_canRemoveLiquidityDualWhenBelowIM() (gas: -79 (-0.023%))
test_addLiquidityDual() (gas: -79 (-0.026%))
test_addLiquidityDual() (gas: -79 (-0.026%))
test_personalizedMaintMargin() (gas: -79 (-0.026%))
test_deleverageZeroSize() (gas: -79 (-0.026%))
test_withBookAndOneAMM_iocPartiallyFilled_notRevert() (gas: -79 (-0.026%))
test_removeLiquidityDualWhenNeutral() (gas: -316 (-0.027%))
test_removeLiquiditySingleWhenNeutral() (gas: -316 (-0.027%))
```

```
test_swapShort() (gas: -79 (-0.027%))
test_swapShort() (gas: -79 (-0.027%))
test_removeLiquidityDualWhenNeutral() (gas: -316 (-0.027%))
test_removeLiquiditySingleWhenNeutral() (gas: -316 (-0.027%))
test_swapLong() (gas: -79 (-0.027%))
test_swapLong() (gas: -79 (-0.027%))
test_forceDeleverage_returnValues() (gas: -79 (-0.027%))
test_orderAndOtc_returnValues() (gas: -177 (-0.028%))
test_otcAndCashTransferMargin() (gas: -237 (-0.029%))
test_basicOTC() (gas: -79 (-0.030%))
test_addThenRemoveLiquiditySingleReturnsSameCash_positiveLong() (gas: -31600 (-0.031%))
test_slippage_addLiquidityDual() (gas: -237 (-0.031%))
test_slippage_addLiquidityDual() (gas: -237 (-0.031%))
test_maintMarginCalculation() (gas: -7900 (-0.032%))
test_addThenRemoveLiquiditySingleReturnsSameCash_positiveLong() (gas: -31600 (-0.032%))
test_personalFeeFactor_otc() (gas: -237 (-0.032%))
test_otcAndCashTransfer() (gas: -158 (-0.033%))
test_canNotAddLiquidityDualWhenBelowIM() (gas: -79 (-0.033%))
test_canNotAddLiquidityDualWhenBelowIM() (gas: -79 (-0.033%))
test_addThenRemoveLiquiditySingleReturnsSameCash_positiveShort() (gas: -31679 (-0.034%))
test_otcAndCashTransfer_returnValues() (gas: -79 (-0.034%))
test_addThenRemoveLiquiditySingleReturnsSameCash_positiveShort() (gas: -31679 (-0.034%))
test_otcMultipleCounters() (gas: -177 (-0.037%))
test_revert_deleverageNotReduceOnly() (gas: -158 (-0.041%))
test_slippage_removeLiquiditySingle() (gas: -474 (-0.041%))
test_slippage_removeLiquiditySingle() (gas: -474 (-0.042%))
test_addThenRemoveLiquidityDual() (gas: -158 (-0.042%))
test_addThenRemoveLiquidityDual() (gas: -158 (-0.042%))
test_canNotOTC() (gas: -79 (-0.053%))
test_canNotRemoveLiquiditySingleWhenBelowIM() (gas: -158 (-0.053%))
test_canNotRemoveLiquiditySingleWhenBelowIM() (gas: -158 (-0.053%))
test_removeLiquiditySingle() (gas: -474 (-0.059%))
test_removeLiquiditySingle() (gas: -474 (-0.059%))
test_slippage_removeLiquidityDual() (gas: -316 (-0.063%))
test_slippage_removeLiquidityDual() (gas: -316 (-0.064%))
Overall gas change: -152970 (-0.002%)
```

Recommendation:

```
+ UserMem memory user = users[userID];
  for (uint256 i = 0; i < OTCs.length; i++) {
    Trade trade = OTCs[i].trade;
    sumPayFee[userID] = sumPayFee[userID].addPayment(
-   _mergePostProcess(users[userID], market, trade, TradeType.OTC)
+   _mergePostProcess(user, market, trade, TradeType.OTC)
    );
    sumPayFee[i] = sumPayFee[i].addPayment(
        _mergePostProcess(users[i], market, trade.opposite(), TradeType.OTC)
    );
  }

- uint256[] memory fees = _calcOtcFees(market, users[userID].addr, OTCs);
+ uint256[] memory fees = _calcOtcFees(market, user.addr, OTCs);
  for (uint256 i = 0; i < OTCs.length; i++) {
    totalFees += fees[i];
-   emit OtcSwap(users[userID].addr, users[i].addr, OTCs[i].trade, fees[i]);
+   emit OtcSwap(user.addr, users[i].addr, OTCs[i].trade, fees[i]);
  }
```

Pendle Finance: N.A. due to significant refactoring.

Spearbit: Acknowledged.

6.3.3 Replace >= with == for loop break condition

Severity: Gas Optimization

Context: RecentTradeRateLib.sol#L46

Description: The loop break condition `curPeriod - i >= RecentTradeRateReadWriteLib.MAX_STORED_PERIODS` can be switched to == instead of >=. As the loop decrements `i` from `curPeriod`, the break will always occur at exactly `i = curPeriod - MAX_STORED_PERIODS`.

Proof of Concept: Gas savings are as follows:

```
test8() (gas: -200 (-0.000%))
test7() (gas: -200 (-0.000%))
test6() (gas: -200 (-0.000%))
test_multipleIntervals() (gas: -408 (-0.015%))
test_multipleIntervals() (gas: -408 (-0.015%))
test_canNotPlaceOrder() (gas: -153 (-0.020%))
test5_ongoing_period() (gas: -200 (-0.033%))
test1() (gas: -200 (-0.035%))
test2() (gas: -203 (-0.036%))
test_revert_check_time_too_far() (gas: -200 (-0.038%))
test_norevert_check_time() (gas: -200 (-0.038%))
test3() (gas: -218 (-0.038%))
test4() (gas: -251 (-0.045%))
Overall gas change: -3155 (-0.000%)
```

Recommendation:

```
- if (curPeriod - i >= RecentTradeRateReadWriteLib.MAX_STORED_PERIODS) break;
+ if (curPeriod - i == RecentTradeRateReadWriteLib.MAX_STORED_PERIODS) break;
```

Pendle Finance: The `RecentTradeRateLib` is removed.

Spearbit: Acknowledged.

6.4 Informational

6.4.1 `memory-safe` annotations

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation:

- ArrayLib.sol#L82-L110: For `LowLevelArrayLib.setLength` might not be memory safe in general, if the provided length is greater than the original length. Although in the current codebase, whenever these functions are used this invariant is indirectly guaranteed. Out of all the call sites the one at `OrderBookUtils.sol#L262` requires more attention, although if this invariant would break, then there would be an earlier out of bound error.
- ArrayLib.sol#L63-L80: `sliceFromTemp` and `restoreSlice` are used together to create a temporary slice and then restore the overwritten array element.

Pendle Finance: Fixed in commit 711a6d1b.

Spearbit: Fix verified.

6.4.2 Make sure the chain used for deployment supports `MCOPY`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Context: ArrayLib.sol#L24-L59

Description: In `ArrayLib`, the functions `extend`, `concat`, and `sliceFrom` use `MCOPY`.

Recommendation: Make sure the chain used for deployment supports `MCOPY`.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.3 Typos, Minor Recommendations

Severity: Informational

Context: FIndexOracle.sol#L124, Tick.sol#L355, Tick.sol#L357-L358, Order.sol#L84

Description/Recommendation:

Typos:

- RecentTradeRateLib.sol#L7: `maner` → `manner`.
- IMarket.sol#L128: `200` → `192`.
- FIndexOracle.sol#L124: The algorithm used actually finds the smallest `k`. So this comment line should read:

```
// find the smallest k such that
```

- Tick.sol#L353: `(has, form)` → `(have, forms)`.
- FIndexOracle.sol#L177: `Use this these of PMath` → `Use this from PMath`.
- Tick.sol#L355: `3000000...000` → `2000000...000`.
- Tick.sol#L357-L358: `ODD_BIT_MASK` → `EVEN_BIT_MASK` since this is the `uint40` with all the even bit indices set $\cdots 0101$.
- MarketOrderAndOtc.sol#L47: `orderAndOtc` has comments for each phase of the function (from Phase 1 to Phase 6), but is missing one for Phase 2. It should say something like `Phase 2: Sweep and calculate initial margins`.
- Order.sol#L84: `(begining, prioirty)` → `(beginning, priority)`.

Minor Recommendations:

- CoreStateUtils.sol#L51: Potential Gas Optimisation if we reuse `$` instead of `_accState(addr)`.
- Tick.sol#L207: Comment out `newActiveTickNonce` because it's unassigned; for consistency.
- ProcessUtils.sol#L32, ProcessUtils.sol#L127:

```
- if (user.part.fTime != 0)
+ if (!user.part.isZero)
```

- CoreOrderUtils.sol#L76: Perhaps a custom error can be used instead of `assert(false)`.
- CreateCompute.sol#L35: Another instance of `assert(false)` that could be replaced with a custom error.
- MarketFactory.sol#L56, MarketSetAndView.sol#L78-L87: Prefer to have the maturity check shifted to `MarketSetAndView` since that's where most of the sanity checks reside.
- MarginManager.sol#L211: For clarity rename `__settleProcessGetIM` to `_settleProcessGetMargin`as the type of margin is determined by `req`. This also applies to `totalIM`in several places when the type of margin it represents can be both.
- MarginManager.sol#L141: To fit the codebase's naming standard, `_processAndCheckIMWeak` should also mention the strict check that happens in the function if the weak check fails (e.g., `_processAndCheckIMWeakOrStrict`).

- MarginManager.sol#L82: `uint256 maturity` → `uint32 maturity`.
- MarginManager.sol#L83: `uint256 latestFTime` → `uint32 latestFTime`.

Pendle Finance: Fixed in commit 711a6d1b.

Spearbit: Fix verified.

6.4.4 Parameter checks are missing during construction or initialisation, or calls to setter functions

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation:

- FIndexOracle.sol#L27-L32: `FIndexOracle.constructor` is missing checks for its input values. For example bound checks for `updatePeriod_`, `maxUpdateDelay_`, `settleFeeRate_`, `maturity_`, whether the `market_` has been created using a desired marker factory and if `permissionController_` is registered in a known address directory contract. Same suggestions apply to `setConfig`.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.5 Tokens with greater than `18` decimals would not be able to be registered in the market hub

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Context: Storage.sol#L129

Description: Tokens with greater than `18` decimals would not be able to be registered in the market hub. This is due to the factor that during the call to `registerToken`, we calculate the `scalingFactor` factor as:

```
uint96 scalingFactor = uint96(10 ** (18 - IERC20Metadata(token).decimals()));
```

and so the subtraction would underflow. Thus tokens such as `NEAR` which has `24` decimals would not be able to be registered. The above `scalingFactor` is used to make sure `accCash[payer]` has `18` decimals precision independent of the token used.

Recommendation: To have the possibility of registering tokens with higher decimals, one instead can store the exponent as `int16` (`scalingFactorExponent`) and depending on the sign of to convert the unscaled amount to the scaled one, use multiplication or division. One caveat of the negative sign would be division might introduce errors in scaled token/cash amounts which might be undesirable so extra care needs to be taken to make sure these divisions are always in favour of the protocol (since the user might be able to abuse the system otherwise). Another solution would be to use a higher precision and make sure the set of tokens that the protocol would like to support in the future do not surpass that max precision/decimals.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.6 Correctness Analysis of `rawCoverLength`

Severity: Informational

Context: Tick.sol#L345-L360

Description: let $n \in \{0, 1, \cdots, 2^{40} - 1\}$ (ie, n is a `uint40`). Assume the binary and quaternary representation of n are given by:

$$n = (b_{39} \cdots b_1 b_0)_2 = (q_{19} \cdots q_1 q_0)_4$$

Letting y to be the number of trailing 1 s of n in binary (not quaternary) representation, we have:

$$n = \cdots b_{y+1}011\cdots1 = N2^{y+1} + (2^y - 1)$$

One can find an $N \in \mathbb{N} \cup 0$ such that the last equality above satisfies:

$$n = \begin{cases} N2^{2k+1} + (2^{2k} - 1) & y = 2k \\ N2^{2k+2} + (2^{2k+1} - 1) & y = 2k+1 \end{cases}$$

or...

$$nX = \begin{cases} 2N4^k + (4^k - 1) & y = 2k \\ N4^{k+1} + (4^k + 4^k - 1) & y = 2k+1 \end{cases}$$

or...

$$n = \begin{cases} (\cdots q_k 33\cdots3)_4 & y = 2k \to q_k \in \{0,2\} \\ (\cdots q_k 33\cdots3)_4 & y = 2k+1 \to q_k = 1 \end{cases}$$

and so k represents the number of trailing 3s of n and 4^k is the value that `rawCoverLength` should return which is either 2^y for an even y or 2^{y-1} for an odd y. Going through the algorithm one can see that initially `res` (r) is calculated as:

$$r = (n+1) \wedge \neg n$$

$$\begin{array}{rccccc} & \cdots & b_{y+1} & 1_y00\cdots0 & (n+1) \\ \wedge & \cdots & \neg b_{y+1} & 1_y00\cdots0 & (\neg n) \\ \hline r = & \cdots & 0 & 1_y00\cdots0 & \end{array}$$

and so $r = 2^y$. Next one checks for the oddness of y:

$$\begin{aligned} \texttt{ODD\_BIT\_MASK} &= \texttt{0x55\_55\_55\_55\_55} \\ &= \texttt{0b0101010101010101010101010101010101010101} \\ &= \sum_{i=0}^{19} 2^{2i} \end{aligned}$$

by checking whether $r \wedge$ `ODD_BIT_MASK` is 0 or not and based on that we divide r by 2.

Edge case: $n = 2^{40} - 1$: In this special case inside the `unchecked` block `nodeId + 1` ($n+1$) wraps around and becomes 0 and thus the function in this case returns 0 instead of the expected value 4^{20}. The following test case fails:

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity ^0.8.28;

import {Test} from "forge-std/Test.sol";

contract CantinaTickLibTest is Test {
    function rawCoverLength(uint40 nodeId) private pure returns (uint40 res) {
        unchecked {
            // wrap-around arithmetic is desired
            res = (nodeId + 1) & ~nodeId; // <---
```

```
        }

        uint40 ODD_BIT_MASK = 0x55_55_55_55_55;
        bool isPowerOf4 = (res & ODD_BIT_MASK) > 0;
        if (!isPowerOf4) res >>= 1;
    }
    function testRawCoverLengthEdge() public {
        uint40 nodeId = type(uint40).max;
        uint40 res = rawCoverLength(nodeId);

        vm.assertEq(res, 4 ** 20); // <--- test fails
    }
}
```

Pendle Finance: Fixed in commit 62018338.

Spearbit: Fix verified.

6.4.7 Custom errors not properly defined in `PMath`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The error messages displayed as a result of reverts in `PMath` are unclear or garbled because the custom errors are not explicitly defined. Instead, only their selectors are used directly in assembly. This makes debugging and error handling difficult.

Proof of Concept:

```
function testRevertMulDownFails() public pure {
  PMath.mulDown(type(uint256).max, 2);
}

function testOverflow() public pure {
  PMath.Int128(type(uint256).max);
}

function testRevertSMulDownFails() public pure {
  PMath.mulDown(type(int256).min, -1);
}

function testRevertDivDownFails() public pure {
  PMath.divDown(type(uint256).max, 1); // Will fail since x > type(uint256).max / ONE
}

function testRevertSDivDownFails() public pure {
  PMath.divDown(type(int256).min, 1); // Will fail since x * ONE overflows
}

function testRevertRawDivUpFails() public pure {
  PMath.rawDivUp(1, 0); // Will fail since dividing by zero
}
```

The following test cases revert with:

```
[FAIL: custom error 0x35278d12] testOverflow() (gas: 214)
[FAIL: |_H}] testRevertDivDownFails() (gas: 261)
[FAIL: custom error 0xbac65e5b] testRevertMulDownFails() (gas: 247)
[FAIL: e$NN] testRevertRawDivUpFails() (gas: 274)
] testRevertSDivDownFails() (gas: 203)
[FAIL: custom error 0xedcd4dd4] testRevertSMulDownFails() (gas: 319)
```

Specifically, `testRevertSDivDownFails` reverts with:

```
[203] Test::testRevertSDivDownFails()
    ← [Revert] \Ct
```

which shows up weirdly on the console.

Recommendation: Explicitly define the custom errors.

```
error MulWadFailed();
error SMulWadFailed();
error DivWadFailed();
error SDivWadFailed();
error DivFailed();
error Overflow();
```

Pendle Finance: The error selector is not important for the contract.

Spearbit: Acknowledged.

6.4.8 `MarkRate` in transient storage is never reset

Severity: Informational

Context: MarketInfoAndState.sol#L174-L178

Description: When the `markRate` is initially determined, it is cached in transient storage without being reset, contrary to the the security recommendations of EIP1153 This effectively caches the `markRate` for the duration of the transaction keeping it the same for the root call frame. The caching is intended to last throughout multiple calls into the `market`, the `marketHub` and even further upstream contracts like the AMM and the router.

At some point however some transaction will update the `markRateOracle` with a new `markRate`. If this is done in a permisionless way an attacker could create a situation where the market is operating with an old `markRate` compared to what is currently in the `markRateOracle`. If there is any other contract relying on the same oracle it would make the market susceptible to price descrepencies which an attacker could abuse. These other contract could be other markets using the same TokenId.

Note that permissionless updating the oracle in this case can take the form of account abstraction where any `tx.origin` can sponsor the transaction on behalf of the oracle's permissioned role or cross chain messaging if supported by the oracle.

Proof of Concept: A possible scenario how this could be exploited is the following. The attacker deploys a custom contract which then:

1. Calls `MarketEntry::getMarkRate()` on the market.
2. Executes the permisionless call to update the oracle.
3. Executes trades on Pendle using the old `markRate` and combining them with calls on other contracts using the same oracle with the new `markRate`.

Recommendation: Add an additional function to reset the transient storage and at the end of the highest level call meant to use the cached `markRate` use this call to reset the cache, thereby forcing a reload on the next call to the market (which under normal circumstances would happen anyway).

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.9 Rounding Directions

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The protocol calculates many different values from different categories. But in general one should make sure that:

value type	rounding direction
fee	↑ overestimate
margin	↑overestimate
cost	↑ overestimate
value	↓ underestimate
payment	↓ underestimate

Also depending on the context, the protocol should perform symmetric rounding towards 0. This for example applies to some cases in "Analysis of `forceDeleverage`".

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.10 Checking the open interest cap for the liquidation flow is not necessary

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: For the `liquidation` flow, let's define:

parameter	description
$\mathcal{OI}$	market's open interest
u_l	`liquidator`
u_v	`violator`
s_{u_l}	signed size of the `liquidator` depending on the context
s_{u_v}	signed size of the `violator` depending on the context
$\blacksquare s$	`sizeToLiquidator`

Then we have:

$$s_{u_l} \rightarrow s_{u_l} + \blacksquare s$$
$$s_{u_v} \rightarrow s_{u_v} - \blacksquare s$$

Moreover due to the `_isReducedOnly` check we know that:

1. $|-\blacksquare s| \leq |s_{u_v}|$.
2. sign($\blacksquare s$) = sign(s_{u_v}) (unless $\blacksquare s = 0$ which is an obvious edge case).

   from 1. and 2. we can deduce that:

3. $|s_{u_v} - \blacksquare s| = |s_{u_v}| - |\blacksquare s| \leq |s_{u_v}|$.

Let's see how $\mathcal{OI}$ gets affected in the liquidation process:

$$\begin{aligned}\mathcal{OI}^{in} = \mathcal{OI} \rightarrow & \mathcal{OI} - |s_{u_l}| + |s_{u_l} + \blacksquare s| \\ \rightarrow & \mathcal{OI} - |s_{u_l}| + |s_{u_l} + \blacksquare s| - |s_{u_v}| + |s_{u_v} - \blacksquare s| \\ = & \mathcal{OI} - |s_{u_l}| + |s_{u_l} + \blacksquare s| - |\blacksquare s| \\ = & \mathcal{OI} + |s_{u_l} + \blacksquare s| - (|s_{u_l}| + |\blacksquare s|) \\ = & \mathcal{OI}^{out} \leq \mathcal{OI}^{in}\end{aligned}$$

The above inequality $\mathcal{OI}^{out} \leq \mathcal{OI}^{in}$ is due to the fact that by triangle inequality we have:

$$|s_{u_l} + \blacksquare s| \leq |s_{u_l}| + |\blacksquare s|$$

and so $|s_{u_l} + \blacksquare s| - (|s_{u_l}| + |\blacksquare s|)$ is a non-positive number.

and so if the open interest cap requirement was satisfied for $ \mathcal{OI}^{in}$ originally it would also be satisfied for $\mathcal{OI}^{out}$ after liquidation.

Recommendation: The line below regarding the `MATURITY_AND_CAP` could be changed to:

```
_writeMarketMem(market, WriteChecks.ONLY_MATURITY);
```

One can keep the open interest cap check at this point as well to potentially catch other inconsistencies that might arise in the codebase.

Footnote: The proof above relies on the facts that:

1. The implementation of `abs` is correct.
2. No unchecked blocks or arithmetic overflow/underflow happens in the calculations.

Pendle Finance: Fixed in commit 711a6d1b.

Spearbit: Fix verified.

6.4.11 Consider initializing `lastTradedTime` as `0` for initial implied rate to be that of seeded tick

Severity: Informational

Context: MarketImpliedRate.sol#L18

Description: The initial values set for the market implied rate is `0` for `_prevOracleRate` and `block.timestamp` for `_lastTradedTime`. Should the rate at the seeded tick be more desirable as the initial oracle rate, one could consider setting `_lastTradedTime` to `0`.

Recommendation:

```
- return from(0, initStruct.window, uint32(block.timestamp), initStruct.seedTradedTick);
+ return from(0, initStruct.window, 0, initStruct.seedTradedTick);
```

Pendle Finance: The `RecentTradeRateLib` is removed.

Spearbit: Acknowledged.

6.4.12 Open interest `OI` differs from conventional definition

Severity: Informational

Context: MarketSetAndView.sol#L96

Description: The market `OI` includes both the long and short sizes, which differs from conventional perps where only 1 side is counted. Hence, it is important for the `OICap` to be set with this variant definition in mind. Otherwise, the `OICap` would be half the expected value.

Recommendation: Clearly document this behaviour; consider adding a `@dev` note in `MarketSetAndView.setMarketConfig()` and / or in `MarketFactory.create()`.

Pendle Finance: We have updated the whitepaper to reflect this.

Spearbit: Acknowledged.

6.4.13 Some `UUPSUpgradeable` inherited contract do not follow `ERC-7201` for storage layout

Severity: Informational

Context: Storage.sol#L32-L43, MarketFactory.sol#L21, MarketFactory.sol#L24

Description: `Storage` contract does not follow `ERC-7201` for storage layout. `Storage` inherits from `UUPSUpgradeable` and so it will sit behind an upgradable proxy. In other contracts which are inherited from `UUPSUpgradeable`, the codebase uses the *Namespaced Storage Layout* of `ERC-7201` to avoid storage layout issues after upgrades.

Recommendation: Use a custom storage root for the storage layout of `Storage` just like other instances of `UUPSUpgradeable` contracts. The same suggestion applies to:

- MarketFactory.sol#L21, MarketFactory.sol#L24

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.14 Typos, Minor Recommendations

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description / Recommendation:

- Typos:
  - EIP712.sol#L3:

```
- determinitic
+ deterministic
```

  - BookAmmSwapBase.sol:

```
- (cash /*totalIM*/, , size) = _MARKET_HUB.settleAllAndGet(amm, GetRequest.ZERO,
↪  amm.marketId());
+ (cash, /*totalIM*/, size) = _MARKET_HUB.settleAllAndGet(amm, GetRequest.ZERO,
↪  amm.marketId());
```

- Minor Recommendations:
  - PendleRoles.sol#L18:

```
- _AMM_ADMIN_ROLE_NAMESPACE = keccak256("AMM_ADMIN");
+ _AMM_ADMIN_ROLE_NAMESPACE = keccak256("AMM_ADMIN_ROLE");
```

    Alternatively, since namespace isn't a role, perhaps it would be good to rename it differently:

```
bytes32 internal constant _FINDEX_ORACLE_UPDATER_ROLE_NAMESPACE =
↪  keccak256("FINDEX_ORACLE_UPDATER_ROLE_NS");
bytes32 internal constant _AMM_ADMIN_ROLE_NAMESPACE = keccak256("AMM_ADMIN_ROLE_NS");
```

    Generally, to follow the pattern `bytes32 internal constant _X = keccak256("X");`.

  - Errors.sol#L60:

```
- error AMMInsufficientCashIn();
+ error AMMCashInExceeded();
```

- BaseAMM.sol#L51: Modifier name is a little generic, isn't descriptive of the functionality.

```
- nonViewModifier
+ onlyRouterAndUpdateOracle // alternative: onlyRouterWithOracleUpdate
```

- slots.sol#L33: looks like that there is a general pattern that storage locations names end with `_LOCATION` and transient storage slots end with `_SLOT`.

```
- MARKET_CACHE_MARK_RATE
+ MARKET_CACHE_MARK_RATE_SLOT
```

- MiscModule.sol#L98-L103: Sanity check that `ammAcc` isn't zero, since it's used as an existence flag in the getter.

```
+ require(!ammAcc.isZero(), Err.AMMCannotBeZero);
```

- TickSweepStateLib.sol#L10-L14: The various tick sweep state stages can be defined as an enum instead of constants.

- Unused Code:
  - IRouterEventsAndTypes.sol#L146-L155: `StopOrder` is not used.
  - PendleRoles.sol#L14: `_AGENT_GUARDIAN_ROLE` is not used.

Pendle Finance: Fixed in commit 711a6d1b.

Spearbit: Fix verified.

6.4.15 Inconsistent access control for `initialize()` functions across different contracts

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The codebase has inconsistent access control for `initialize()` functions across different contracts:

contract	A/C	implementation
AdminModule	No	`... initialize() ... { __UUPSUpgradeable_init(); }`
MarketHub	No	`... initialize() ... { __Storage_init(); __UUPSUpgradeable_init(); }`
AMMFactory	No	`... initialize() ... { __UUPSUpgradeable_init(); }`
Explorer	No	`... initialize() ... { __UUPSUpgradeable_init(); }`
MarketFactory	Yes	`... initialize() ... onlyMasterAdmin { __UUPSUpgradeable_init(); marketNonce = 1; }`

Note:

1. Currently `__UUPSUpgradeable_init` is a no-op.
2. `__Storage_init();` for `MarketHub` exists so that the master admin would not accidentally register a native token. The scaling factor is set to 0 for the native token.
3. `marketNonce = 1;` for `MarketFactory` exists so that the created markets by the master admin start with market id 1. `MarketIdLib.ZERO` is reserved so that in `__settleProcessGetIM` one could set `idToSkip` or `idToGetSize` to that special market id value. So it is very important that `initialize()` is called before `create` in this context. This could have been simplified if we performed the following diff:

```
  diff --git a/contracts/factory/MarketFactory.sol b/contracts/factory/MarketFactory.sol
  index ccee192..fe91a0f 100644
-  -- a/contracts/factory/MarketFactory.sol
+  ++ b/contracts/factory/MarketFactory.sol
  @@ -32,11 +32,8 @@ contract MarketFactory is IMarketFactory, PendleRolesPlugin, UUPSUpgradeable
   ↪  {
          IMPLEMENTATION = _implementation;
      }

-      function initialize() external initializer onlyMasterAdmin {
+      function initialize() external initializer {
          __UUPSUpgradeable_init();
-
-          // nonce starts from 1
-          marketNonce = 1;
      }

      // solhint-disable-next-line ordering, no-empty-blocks
  @@ -55,7 +52,7 @@ contract MarketFactory is IMarketFactory, PendleRolesPlugin, UUPSUpgradeable
   ↪  {
          require(IMarketHub(MARKET_HUB).tokenData(tokenId).token != address(0),
          ↪  Err.InvalidTokenId());
          require(maturity > uint32(block.timestamp), Err.InvalidMaturity());

-          MarketId newMarketId = MarketId.wrap(marketNonce);
+          MarketId newMarketId = MarketId.wrap(++marketNonce);
          MarketImmutableDataStruct memory immData = MarketImmutableDataStruct({
              name: name,
              symbol: symbol,
  @@ -80,7 +77,5 @@ contract MarketFactory is IMarketFactory, PendleRolesPlugin, UUPSUpgradeable
   ↪  {
          assert(allMarkets.add(newMarket));

          emit MarketCreated(newMarket, immData, config);
-
-          marketNonce++;
      }
  }
```

Recommendation: It would be great to unify the access/control for the above endpoints.

Pendle Finance: Fixed in commit 11913c82.

Spearbit: Fix verified.

6.4.16 Discrepancies between factory contracts

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: There are a few discrepancies between factory contracts `MarketFactory` and `AMMFactory`. Both factories store the implementation of the contracts that they would deploy as immutable parameters. But there are a few minor differences:

1. `MarketFactory` keeps track of its deployed market contracts as a set in `allMarkets` but `AMMFactory` does not.
2. Market ids are automatically set starting from 1 in `MarketFactory` but in `AMMFactory` the `ammId` is provided manually by the master admin to the `create` endpoint.

3. In `MarketFactory`, the common `MARKET_HUB` contract is stored as an immutable parameter which does not need to be provided each time for market deployments. But in `AMMFactory`, the `router` contract is provided by the master admin manually each time a new AMM contract needs to be created.
4. `MarketFactory.create` can only be called by the master admin, but **anyone** can call `AMMFactory.create`.
5. Deployed markets are wrapped with a `TransparentUpgradeableProxy` and thus are upgradable ver as deployed `AMM`s are not upgradable.

Recommendation: It would be nice to in `AMMFactory`:

1. Keep track of a set of deployed AMMs.
2. Provided `ammId` automatically.
3. If the `router` contract is going to be the same among all deployed AMMs, it would be best to have it as an immutable parameter.
4. It would be best to have the same access/control for both `MarketFactory.create` and `AMMFactory.create`.
5. Perhaps the deployed `AMM`s can also be upgradable. Moreover if the expected number of deployed markets and AMMs is going to be a big number, it might make sense to use a beacon proxy pattern (or a hybrid of that).

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

6.4.17 Suggestions regarding `PendleAccessController`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation:

1. `_MASTER_ADMIN_ROLE` is the same as `AccessControlUpgradeable.DEFAULT_ADMIN_ROLE` which is public constant and the default admin role of all roles.
2. The lines PendleAccessController.sol#L22, PendleAccessController.sol#L26, PendleAccessController.sol#L31:

   ```
   require(hasRole(_MASTER_ADMIN_ROLE, msg.sender), Err.Unauthorized());
   ```

   can be changed to:

   ```
   _checkRole(_MASTER_ADMIN_ROLE); // or
   _checkRole(DEFAULT_ADMIN_ROLE)
   ```

   or one can instead add the `modifier`, `onlyRole(_MASTER_ADMIN_ROLE)`.
3. Currently, the admin role of any roles can not be changed from `_MASTER_ADMIN_ROLE` since `_setRoleAdmin` is not externally exposed, so `grantFIndexOracleUpdaterRole` and `grantAMMManagerRole` can stay intact.
4. `PendleAccessController` has special external endpoints for granting roles for `fIndexOracleUpdater` of an `fIndexOracle` or admin role for an `AMM`, but the same granting endpoints are missing for the `internal` constant variables:
   - `_DELEVERAGER_ROLE`.
   - `_MARKET_ADMIN_ROLE`.
   - `_DIRECT_MARKET_HUB_ROLE`.

   perhaps special granting endpoints should be defined for these roles as well.
5. Any account with `_MARKET_ADMIN_ROLE` can be an admin for **all** deployed markets. This is in contrast to `AMM` admin roles that are specific to each `AMM` address. Perhaps the market admin roles can also be granular.

Pendle Finance: Acknowledged. This finding is about the old `AccessController`.

Spearbit: Acknowledged.

6.4.18 Boundary checks missing when setting `feeRate` and `oracleImpliedRateWindow`

Severity: Informational

Context: BaseAMM.sol#L37-L38

Description: In `BaseAMM` boundary checks missing when setting `feeRate` and `oracleImpliedRateWindow`. For example, a similar ramping window variable for deployed markets is only allowed to be set in a specific range

Recommendation: Analysis and perform required boundary checks for these variables.

Pendle Finance: All missing boundary checks are intentional since we not sure if we want to embed them in contracts (they must have all been verified thoroughly off-chain) but after discussion we will add some critical checks on contracts.

Spearbit: Acknowledged.

7 Pendle Core v3 Fix Review 1

7.1 Medium Risk

7.1.1 Incorrect bound check when updating the `findex`

Severity: Medium Risk

Context: FIndexOracle.sol#L52-L53

Description: When `updateFloatingRate` by a `keeper` the value provided as `floatingRate` is actually a time-weighted value $r_s \blacksquare t_s$ (here s indexes the s-th update and $\blacksquare t_s$ is the delta time between the s-th update and its previous one) and therefore when one accumulates these values into `_latestFIndex` one gets (`floatingIndex`):

$$\texttt{floatingIndex} = i_f(t) = \sum_s r_s \blacksquare t_s$$

Above r_s corresponds to `_latestAnnualizedRate` and the same value is used as settlement rate in the **Proof for Boros** paper section 10. In this section all the bounds rely on $|r_s - r_m|$ where r_m is the mark rate. But here the max deviation for the bound is performed using the following inequality:

$$|r_m - r_s \blacksquare t_s| \leq \sigma$$

Recommendation: The above error possibly might have been caused due to the naming for `floatingRate` parameter. It would be best to rename all instances of this value to `floatingIndexDelta` and update the inequality check so that:

$$|r_m - r_s| \leq \sigma$$

in other words roughly (further gas optimisations can be applied):

```
require(
   (markRate - _latestAnnualizedRate).abs() <= params.maxFRateDeviation,
   Err.FIndexDeviationTooHigh()
);
```

Actually it would make more sense to bound:

$$\frac{|r_m - r_s|}{r_m^+} \leq \sigma$$

which corresponds to something like:

```
require(
   (markRate - _latestAnnualizedRate).abs() <=
   params.maxFRateDeviation.mulDown( // need to double-check whether to use `mulDown` or a different
   ↪  variation due to precision of the factors involved.
      markRate.abs().max(k_iThresh) // `k_iThresh` needs to be fetched from the market
   ),
   Err.FIndexDeviationTooHigh()
);
```

One can even take this further to bound the settlement health jump $D_{s,i}(r_m, r_s)$:

$$D_{s,i}(r_m, r_s) = \frac{\blacksquare t}{(t_s - \blacksquare t)^+} \cdot \frac{|r_s - r_m|}{\lambda_{MM} r_m^+ \alpha} \leq \sigma$$

Pendle Finance: The floating rate bound check is removed. The goal is to reduce redundant circular dependencies. We will verify the trustworthiness of the data reported by Chainlink off-chain before updating on-chain.

Spearbit: The check in the finding and related parameters are removed from the codebase in commit e2aa73d0. Perhaps these accounting are performed off-chain now by Pendle.

7.2 Gas Optimization

7.2.1 Decrementing length first is slightly more gas-efficient

Severity: Gas Optimization

Context: CoreOrderUtils.sol#L123-L124

Description: Rather than calling `.dec()` twice, by swapping lines, it becomes more gas efficient by calling `length.dec()` only once and reusing the result.

```
test_maxOpenOrders() (gas: -8 (-0.000%))
test_pmUpdateWhenRemovePurgeSettleOrders() (gas: -8 (-0.000%))
test_orderAndOtc_returnValues() (gas: -8 (-0.001%))
test_forceCancel() (gas: -6 (-0.001%))
test_cancelOrderById() (gas: -8 (-0.001%))
test_cancel_returnValues() (gas: -8 (-0.001%))
test_orderAndOtc_cancelOrders() (gas: -8 (-0.001%))
test_cancelOrderPendingSettle() (gas: -8 (-0.001%))
test_partialMatchBasic() (gas: -8 (-0.001%))
test_modifyOrder() (gas: -16 (-0.001%))
test_getOrder() (gas: -8 (-0.002%))
test_cancelOrderDuplicate() (gas: -8 (-0.002%))
test_bulkModify() (gas: -16 (-0.003%))
test_cancelOrderWhenBadDebt() (gas: -24 (-0.003%))
test_otcRateOutOfRange() (gas: 0 (NaN%))
test_approveAgentWithRoot() (gas: 0 (NaN%))
test_defaultAccManager() (gas: 0 (NaN%))
Overall gas change: -142 (-0.000%)
```

Recommendation:

```
- ids[i] = ids[length.dec()];
- length = length.dec();
+ length = length.dec();
+ ids[i] = ids[length];
```

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

7.2.2 Redundant else case for `pmTotalLong` / `pmTotalShort` instantiation

Severity: Gas Optimization

Context: MarginViewUtils.sol#L74

Description: The else case when instantiating `pmTotalLong` / `pmTotalShort` is redundant because it's setting to the same default value of 0.

```
test_liquidateMarketAfterMMSwitch() (gas: -64 (-0.000%))
test_liquidateMarketAfterMMSwitch() (gas: -64 (-0.000%))
test_enterThenArbitrageThenCloseIsolatedPosition() (gas: -14 (-0.000%))
test_worstClosingOrder() (gas: -128 (-0.000%))
test_treasury_exoticDecimals() (gas: -14 (-0.000%))
test_ok_sortedLimitTicks() (gas: -14 (-0.000%))
test_canTradeOnOtherMarkets() (gas: -14 (-0.000%))
test_closingOrdersCheck_signedSizeBefore() (gas: 4 (0.000%))
test_placeOrders() (gas: -14 (-0.001%))
test_partialMatchBothSides() (gas: -10 (-0.001%))
test_closingOrdersCheck() (gas: -84 (-0.001%))
test_withBookAndOneAMM_numTicksToTryAtOnce_short() (gas: -1400 (-0.001%))
test_maxOpenOrders() (gas: -14 (-0.001%))
test_cloCheck() (gas: -12 (-0.001%))
test_withBookAndOneAMM_numTicksToTryAtOnce_long() (gas: -1400 (-0.001%))
test_arbitrage_negativeProfit() (gas: -14 (-0.001%))
test_fullLiquidation() (gas: -14 (-0.001%))
test_fullLiquidation() (gas: -14 (-0.001%))
test_liquidate_returnValues() (gas: -10 (-0.001%))
test_withBookAndOneAMM_otcFeeDiscount() (gas: -14 (-0.001%))
test_withBookAndOneAMM_takerFeeDiscount() (gas: -14 (-0.001%))
test_arbitrage_isolatedMarket() (gas: -102 (-0.001%))
test_openInterestUpdate() (gas: -14 (-0.001%))
test_pmCombination_positivePositionSize() (gas: 80 (0.001%))
test_bulkOrders_longSide() (gas: -14 (-0.001%))
test_bulkOrders_shortSide() (gas: -14 (-0.001%))
test_closingOrdersCheck() (gas: -184 (-0.001%))
test_cancelOrderWhenBadDebt() (gas: -12 (-0.001%))
test_getOIAndPendingSizes() (gas: -24 (-0.001%))
test_liquidatorClosingOrdersCheck() (gas: -64 (-0.002%))
test_placeSingleOrder_longSide() (gas: -14 (-0.002%))
test_placeSingleOrder_shortSide() (gas: -14 (-0.002%))
test_getUserInfo() (gas: -14 (-0.002%))
test_partialMatchSettleImmediately() (gas: -14 (-0.002%))
test_withBookAndOneAMM_swapToMinRate() (gas: -14 (-0.002%))
test_gtcPartiallyFilled() (gas: -14 (-0.002%))
test_pmCombination_negativePositionSize() (gas: 124 (0.002%))
test_arbitrage_justEnoughWholeTick() (gas: -14 (-0.002%))
test_hardOICap() (gas: -24 (-0.002%))
test_withBookAndOneAMM_swapToMaxRate() (gas: -14 (-0.002%))
test_withBookAndOneAMM_bookBetter_short() (gas: -14 (-0.002%))
test_withBookAndOneAMM_ammBetter_long() (gas: -14 (-0.002%))
test_withBookAndOneAMM_ammBetter_short() (gas: -14 (-0.002%))
```

```
test_withBookAndOneAMM_bookBetter_long() (gas: -14 (-0.002%))
test_personalDiscRates_taker() (gas: -14 (-0.002%))
test_orderAndOtc_deferredPartialMatch() (gas: -14 (-0.002%))
test_settleAllAndGet_returnValues() (gas: -14 (-0.002%))
test_forceCancelAllRiskyUser() (gas: -14 (-0.002%))
test_partialMatchBasic() (gas: -14 (-0.002%))
test_aloFilledAny() (gas: -14 (-0.002%))
test_withBookAndOneAMM_notMatchShortLimitOrder() (gas: -14 (-0.002%))
test_withBookAndOneAMM_notMatchLongLimitOrder() (gas: -14 (-0.002%))
test_liquidateMultipleMarkets() (gas: -96 (-0.002%))
test_liquidateMultipleMarkets() (gas: -96 (-0.002%))
test_getMarketInfo() (gas: -14 (-0.002%))
test_withBookAndOneAMM_fees() (gas: -14 (-0.002%))
test_withBookAndOneAMM_justEnoughWholeTick() (gas: -14 (-0.002%))
test_forceDeleverageBadDebtMultipleMarkets() (gas: -96 (-0.002%))
test_withBookAndOneAMM_notMatchAMM_short() (gas: -14 (-0.002%))
test_withBookAndOneAMM_notMatchAMM_long() (gas: -14 (-0.002%))
test_modifyOrder() (gas: -26 (-0.002%))
test_orderAndOtc_returnValues() (gas: -14 (-0.002%))
test_iocPartiallyFilled() (gas: -14 (-0.002%))
test_limitOrderBound() (gas: -168 (-0.002%))
test_fokFullyFilled() (gas: -14 (-0.002%))
test_iocFullyFilled() (gas: -14 (-0.002%))
test_gtcFullyFilled() (gas: -14 (-0.002%))
test_revert_selfMatch() (gas: -14 (-0.002%))
test_withdrawTreasury() (gas: -14 (-0.002%))
test_arbitrage_long() (gas: -70 (-0.003%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -64 (-0.003%))
test_arbitrage_minSpread_maxProfit() (gas: -70 (-0.003%))
test_forcePurgeOobOrders_Long() (gas: -42 (-0.003%))
test_forcePurgeOobOrders_Short() (gas: -42 (-0.003%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -64 (-0.003%))
test_upfrontCost() (gas: -28 (-0.003%))
test_getOrder() (gas: -14 (-0.003%))
test_slippage_removeLiquiditySingle() (gas: -32 (-0.003%))
test_simulateTransfer() (gas: -14 (-0.003%))
test_slippage_removeLiquiditySingle() (gas: -32 (-0.003%))
test_withBookAndOneAMM_shortMultipleTicks() (gas: -28 (-0.003%))
test_withBookAndOneAMM_longMultipleTicks() (gas: -28 (-0.003%))
test_tryAggregate_notRequireSuccess() (gas: -14 (-0.003%))
test_tryAggregate_requireSuccess() (gas: -14 (-0.003%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -64 (-0.003%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -64 (-0.003%))
test_fokPartiallyFilled() (gas: -14 (-0.003%))
test_cancelOrderCheckClosing() (gas: -112 (-0.003%))
test_orderAndOtc() (gas: -14 (-0.003%))
test_settleAndGet_returnValues() (gas: -28 (-0.003%))
test_basicPlaceAndMatchOrder() (gas: -28 (-0.003%))
test_getAllOpenOrders() (gas: -24 (-0.003%))
test_maxInitialMarginViaLimitOrder() (gas: -56 (-0.003%))
test_lastTradedRateBound() (gas: -168 (-0.003%))
test_canNotAddLiquiditySingleWhenBelowIM() (gas: -32 (-0.003%))
test_simulatePlaceOrder() (gas: -14 (-0.003%))
test_cancelOrderDuplicate() (gas: -14 (-0.003%))
test_canNotAddLiquiditySingleWhenBelowIM() (gas: -32 (-0.004%))
test_cancelOrderPendingSettle() (gas: -28 (-0.004%))
test_impliedRate_basic() (gas: -56 (-0.004%))
test_orderAndOtc_returnValues() (gas: -56 (-0.004%))
test_isolatedPlaceOrder() (gas: -14 (-0.004%))
test_slippage_addLiquiditySingle() (gas: -64 (-0.004%))
test_slippage_addLiquiditySingle() (gas: -64 (-0.004%))
test_revert_addLiquiditySingle_insufficientCashIn() (gas: -32 (-0.004%))
```

```
test_cancelOrderById() (gas: -42 (-0.004%))
test_forceCancel() (gas: -34 (-0.004%))
test_forcePurgeOobOrders_returnValues() (gas: -84 (-0.004%))
test_oracleUnset() (gas: -42 (-0.004%))
test_setDeferredImpliedRateOracleDuration() (gas: -42 (-0.004%))
test_purgeOobOrdersNegativeMarkRate() (gas: -28 (-0.004%))
test_settleWhenExitMarket() (gas: -42 (-0.004%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -186 (-0.005%))
test_addLiquiditySingle_bigCash_positiveShort() (gas: -186 (-0.005%))
test_addLiquiditySingleWhenNeutral() (gas: -32 (-0.005%))
test_addLiquiditySingleWhenNeutral() (gas: -32 (-0.005%))
test_addLiquiditySingle_justEnoughWholeTick() (gas: -46 (-0.005%))
test_addLiquiditySingle_justEnoughWholeTick() (gas: -46 (-0.005%))
test_exitMarkets() (gas: -46 (-0.005%))
test_minCashForAccounts() (gas: -32 (-0.005%))
test_cancel_returnValues() (gas: -42 (-0.005%))
test_getUserInfoAfterPlaceOrder() (gas: -46 (-0.005%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -186 (-0.005%))
test_addLiquiditySingle_bigCash_positiveLong() (gas: -186 (-0.005%))
test_removeLiquidityDualWhenNeutral() (gas: -64 (-0.005%))
test_removeLiquiditySingleWhenNeutral() (gas: -64 (-0.005%))
test_orderAndOtc_cancelOrders() (gas: -42 (-0.005%))
test_addLiquidityDualWhenAMMSizeVerySmall() (gas: -32 (-0.005%))
test_addLiquidityDualWhenNeutral() (gas: -32 (-0.005%))
test_removeLiquidityDualWhenNeutral() (gas: -64 (-0.005%))
test_removeLiquiditySingleWhenNeutral() (gas: -64 (-0.005%))
test_addLiquidityDualWhenAMMSizeVerySmall() (gas: -32 (-0.005%))
test_addLiquidityDualWhenNeutral() (gas: -32 (-0.005%))
test_multipleIntervals() (gas: -140 (-0.005%))
test_multipleIntervals() (gas: -140 (-0.005%))
test_marketEntranceFee() (gas: -32 (-0.006%))
test_pmCombination_zeroPositionSize() (gas: -90 (-0.006%))
test_batchForcePurgeOobOrdersLong() (gas: -280 (-0.006%))
test_personalizedInitialMargin() (gas: -56 (-0.007%))
test_bulkModify() (gas: -42 (-0.007%))
test_arbitrage_short() (gas: -134 (-0.007%))
test_removeLiquiditySingle() (gas: -154 (-0.007%))
test_removeLiquiditySingle() (gas: -154 (-0.007%))
test_getMarketOrderBook() (gas: -42 (-0.007%))
test_getNextNTicksLong() (gas: -280 (-0.008%))
test_getNextNTicksShort() (gas: -280 (-0.008%))
test_pmUpdateWhenRemovePurgeSettleOrders() (gas: -162 (-0.008%))
test_settleExcept() (gas: -138 (-0.008%))
test_getTickSumSize() (gas: -84 (-0.009%))
test_vaultTransfer() (gas: -32 (-0.009%))
test_matchManyShortOrders() (gas: -504 (-0.010%))
test_matchManyLongOrders() (gas: -504 (-0.010%))
test_maxOpenOrders() (gas: -308 (-0.010%))
test_payTreasury() (gas: -32 (-0.010%))
test_getUserInfoAfterBulkCancels() (gas: -60 (-0.011%))
test_cancelOrderCheckClosing() (gas: -74 (-0.011%))
test_otcCashTransfer() (gas: -64 (-0.012%))
test_cancelAllOrders() (gas: -280 (-0.012%))
test_enterMultipleMarkets() (gas: -32 (-0.014%))
test_subaccountTransferIsolated() (gas: -32 (-0.016%))
test_subaccountTransferCross() (gas: -32 (-0.016%))
test_cacheMarkRate() (gas: -64 (-0.029%))
test_bench() (gas: -1924 (-0.039%))
test_otcRateOutOfRange() (gas: 0 (NaN%))
test_approveAgentWithRoot() (gas: 0 (NaN%))
test_defaultAccManager() (gas: 0 (NaN%))
```

```
Overall gas change: -13990 (-0.000%)
```

Recommendation:

```
- else pmTotalLong = pmLong > pmAlone ? pmLong - pmAlone : 0;
+ else if (pmLong > pmAlone) pmTotalLong = pmLong - pmAlone;

- else pmTotalShort = pmShort > pmAlone ? pmShort - pmAlone : 0;
+ else if (pmShort > pmAlone) pmTotalShort = pmShort - pmAlone;
```

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

7.3 Informational

7.3.1 Minor Recommendations

Severity: Informational

Context: CoreStateUtils.sol#L38, CoreStateUtils.sol#L65, MarginManager.sol#L96-L100, Storage.sol#L262

Description And Recommendation:

- MarginManager.sol#L96-100: Consider swapping lines to comply with the checks-interactions-effects pattern.

```
+ uint256 userUnscaled = user.unscaled;
+ user.unscaled = 0;
+  _withdrawal[root][tokenId] = user;

- IERC20(data.token).safeTransfer(root, user.unscaled);
- emit VaultWithdrawalFinalized(root, tokenId, user.unscaled);
+ IERC20(data.token).safeTransfer(root, userUnscaled);
+ emit VaultWithdrawalFinalized(root, tokenId, userUnscaled);
```

- Storage.sol#L262: For better clarity, consider renaming `entranceFees` to `entranceFeesScaled`.
- CoreStateUtils.sol#L65: Would be good to add a bit of elaboration on why `longIds` and `shortIds` can't be relied: because they're not read.

```
- // must not rely on longIds & shortIds in this branch
+ // cannot rely on longIds & shortIds as they are empty (storage reads skipped) when
↪  shortcutted
```

- CoreStateUtils.sol#L38: Consider renaming `_initUser()` to `_initUserWithSettlement()` or `_initUserAndSettle()` to give some indication that settlement is performed as well.

Pendle Finance: Fixed in commit a905b378.

Spearbit: Fix verified.

8 Pendle Core v3 Fix Review 2

8.1 Medium Risk

8.1.1 `_isEnoughIMStrict` might not be strong enough

Severity: Medium Risk

Context: MarginManager.sol#L229-L232

Description: In the case that `critHR` H_f is greater than 1 wad. The check in `_isEnoughIMStrict` is not strong enough to guarantee the health ratio would be not less than H_f. Based on the Boros research paper and its Finding 1 we know that (1st inequality comes from the strict `IM` check):

$$v_u(t) \geq \mathcal{IM}_u(t) \geq \frac{k^u_{IM}(t)}{k^u_{MM}(t)} \mathcal{MM}_u(t)$$

so to guarantee that the strict `IM` check would imply the health ratio is not less than H_f, one needs to have:

1. At all times $\frac{k^u_{IM}(t)}{k^u_{MM}(t)} \geq 1$.
2. $v_u(t) \geq \mathcal{IM}_u(t) \cdot \max(1, H_f, H_c)$ (added H_c to the maximum since this is also the value used in example in the boros research paper).

Recommendation:

1. The admins needs to make sure $\frac{k^u_{IM}(t)}{k^u_{MM}(t)} \geq 1$ is always satisfied for all users.
2. Change the `_isEnoughIMStrict` implementation to check:

   ```
   return totalValue + cash >= totalMargin.Int().mulCeil(IONE.max(critHR).max(riskyThresHR))
   ```

   or the market managers can make sure to guarantee that $\frac{k^u_{IM}(t)}{k^u_{MM}(t)} \geq \max(1, H_f, H_c)$.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.2 Low Risk

8.2.1 `rawDivCeil` and `rawDivFloor` return incorrect values for ratios close to `0`

Severity: Low Risk

Context: PMath.sol#L111-L135

Description: Note that:

$$\texttt{sdiv}(x, d) = sign(x/d) \left\lfloor \frac{|x|}{|d|} \right\rfloor$$

and thus `sdiv` rounds towards 0. Thus `rawDivCeil` needs to add 1 to the result when the division is not whole in the region $[0, \infty)$. But `rawDivCeil` uses the following condition to add 1 to the result if necessary:

```
if sgt(z, 0) { /*...*/ }
```

And thus the region $\frac{x}{d} \in (0, 1)$ is neglected. For this region we would have $\texttt{sdiv}(x, d) = 0$. `rawDivFloor` needs to subtract 1 from the result when the division is not whole in the region $(-\infty, 0]$. But `rawDivFloor` uses the following condition to subtract 1 from the result if necessary:

```
if slt(z, 0) { /*...*/ }
```

And thus the region $\frac{x}{d} \in (-1, 0)$ is neglected. For this region we would have $\texttt{sdiv}(x, d) = 0$.

Proof of Concept: In the test suite repo add this file `test/lib/math/PMath.t.sol`:

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity ^0.8.28;

import {PMath} from "boros/lib/math/PMath.sol";

import {Test} from "forge-std/Test.sol";
```

```
contract PMathTest is Test {

    function test_rawDivXX_PositiveRatioLessThanOne(int256 x, int256 d) public {
        // 0 < x/d < 1
        vm.assume(PMath.sign(x) == PMath.sign(d));
        vm.assume(PMath.abs(x) < PMath.abs(d));
        vm.assume(x != 0);

        int256 z = PMath.rawDivCeil(x, d);
        assertEq(z, 1, "rawDivCeil for 0 < x/d < 1 failed");

        z = PMath.rawDivFloor(x, d);
        assertEq(z, 0, "rawDivFloor for 0 < x/d < 1 failed");
    }

    function test_rawDivXX_NegativeRatioGreaterThanNegativeOne(int256 x, int256 d) public {
        // -1 < x/d < 0
        vm.assume(PMath.sign(x) != PMath.sign(d));
        vm.assume(PMath.abs(x) < PMath.abs(d));
        vm.assume(x != 0);

        int256 z = PMath.rawDivCeil(x, d);
        assertEq(z, 0, "rawDivCeil for -1 < x/d < 0 failed");

        z = PMath.rawDivFloor(x, d);
        assertEq(z, -1, "rawDivFloor for -1 < x/d < 0 failed");
    }
}
```

and run:

```
forge test --mc PMathTest
```

Recommendation: Change the conditions to:

```
-  -- a/contracts/lib/math/PMath.sol
+  ++ b/contracts/lib/math/PMath.sol
  @@ -115,7 +115,7 @@ library PMath {
                      revert(0x1c, 0x04)
                  }
                  z := sdiv(x, d)
-                 if sgt(z, 0) {
+                 if iszero(xor(shr(255, x), shr(255, d))) {
                      z := add(z, iszero(iszero(smod(x, d))))
                  }
              }
  @@ -128,7 +128,7 @@ library PMath {
                      revert(0x1c, 0x04)
                  }
                  z := sdiv(x, d)
-                 if slt(z, 0) {
+                 if xor(shr(255, x), shr(255, d)) {
                      z := sub(z, iszero(iszero(smod(x, d))))
                  }
              }
```

Also add test cases to make sure the regions close to 0 return the correct values for these functions.

Pendle Finance: Fixed in commit e2aa73d0.

Spearbit: Fix verified.

8.2.2 Rounding directions are not applied correctly in `_calcPM` (part 1)

Severity: Low Risk

Context: MarginViewUtils.sol#L64-L87

Description/Recommendation: In the code snippet below, it is highlighted which rounding directions need to be applied vs the current one:

```
// Based on where `pmAlone` is used, it would either need to be rounded up or down.
// Currently it is rounded down.
uint256 pmAlone = __calcPMFromRate(S.abs(), market.rMark, market.k_iThresh);

(uint256 B_Long, uint256 pmLong) = (user.pmData.sumLongSize, user.pmData.sumLongPM);
uint256 pmTotalLong;

if (S < 0 && B_Long <= S.abs()) pmTotalLong = 0;
else {
    if (S > 0) pmTotalLong = pmLong + pmAlone; //  `pmAlone` needs to be rounded up. Currently rounded
    ↪  down
    else pmTotalLong = pmLong > pmAlone ? pmLong - pmAlone : 0; //  `pmAlone` needs to be rounded down.
    ↪  Currently rounded down
}

(uint256 B_Short, uint256 pmShort) = (user.pmData.sumShortSize, user.pmData.sumShortPM);
uint256 pmTotalShort;

if (S > 0 && B_Short <= S.abs()) pmTotalShort = 0;
else {
    if (S < 0) pmTotalShort = pmShort + pmAlone; //  `pmAlone` needs to be rounded up. Currently rounded
    ↪  down
    else pmTotalShort = pmShort > pmAlone ? pmShort - pmAlone : 0; //  `pmAlone` needs to be rounded
    ↪  down. Currently rounded down
}
```

1. In general one needs to make sure the value returned by `_calcPM(...)` is overestimated.
2. The accumulated values of `user.pmData.sumLongPM`, `user.pmData.sumShortPM` in memory or storage would always need to round up.
3. 2 variants of `__calcPMFromRate` needs to be defined and depending on the call site the variant with the correct rounding direction needs to be used.

Pendle Finance: Acknowledged. Won't fix as we will scale all to 1e18 to avoid having to be too strict with rounding.

Spearbit: Acknowledged.

8.2.3 Rounding directions are not applied correctly in `_calcMM` (part 2)

Severity: Low Risk

Context: MarginViewUtils.sol#L56

Description/Recommendation: The calculated `PM` value needs to be rounded up in `_calcMM`.

Pendle Finance: Acknowledged. Won't fix as we will scale all to 1e18 to avoid having to be too strict with rounding.

Spearbit: Acknowledged.

8.2.4 Correct rounding direction needs to be used when accumulating `pmData` in memory or storage

Severity: Low Risk

Context: CoreStateUtils.sol#L259, MarginViewUtils.sol#L89-L105, PendingOIPureUtils.sol#L40, PendingOIPure-Utils.sol#L48, PendingOIPureUtils.sol#L61, ProcessMergeUtils.sol#L68

Description/Recommendation: In general when adding to `pmData` the delta accumulated needs to be rounded up and when subtracting from `pmData` the delta subtracted needs to be rounded down. This would guarantee that the initial margin checks are stricter.

location	desired rounding direction	current rounding direction	function
`_squashPartial`	up	down	`_calcPMFromFill`
`_updateOIAndPMOnSwept`	down	down	`_calcPMFromFill`
`_mergePartialFillAft`	down	down	`_calcPMFromFill`
`_updatePMOnAdd`	up	down	`_calcPMFromTick`
`_updatePMOnRemove`	down	down	`_calcPMFromTick`

Pendle Finance: Acknowledged. Won't fix as we will scale all to 1e18 to avoid having to be too strict with rounding.

Spearbit: Acknowledged.

8.2.5 `_calcRateBound` needs to return bounds that are tighter

Severity: Low Risk

Context: MarginViewUtils.sol#L222-L247

Description/Recommendation: `_calcRateBound` needs to return bounds that are tighter when:

- Purging out of bound orders the bounds returned must be tighter so that the areas of ticks purged would be larger.
- Checking the margin the bounds for worst rates need to be tighter.

location	bounds need to be
`_checkMargin`	tighter
`forcePurgeOobOrders`	tighter
`forcePurgeOobOrders`	tighter

Note that the value of first input `rMark` fed into `__calcRateBoundPositive` is always non-negative. Based on the **boros research** paper and also Finding #96 of the paper we know that:

- All the `_ctx().loUpperSlopeBase1e4`, `_ctx().loLowerSlopeBase1e4`, `_ctx().loUpperConstBase1e4` are positive.
- Except `_ctx().loLowerConstBase1e4` (which if the constants picked from the paper or the finding #96 then this value should be the opposite of `_ctx().loUpperConstBase1e4`).

In these scenarios the rounding directions are not tighter:

1. If `market.rMark >= market.k_iThresh` ($I \leq r_m(t)$)and `side == Side.SHORT`.
2. If `-` `market.rMark >= market.k_iThresh` ($r_m(t) \leq -I$) and `side == Side.LONG` (original side).

In general depending on the conditions one needs to use 2 different variant of `mulBase1e4` and round in a direction that would make the bounds tighter.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

Related:

- Research Finding 97.
- Research Finding 96.

Also based on the above research findings the current functions used for calculating the bounds are incorrect.

8.2.6 `checkRateInBound` is only performed on the new set of orders getting added to the orderbook

Severity: Low Risk

Context: MarginViewUtils.sol#L121-L126

Description: `checkRateInBound` is only performed on the new set of orders getting added to the orderbook. This is one of the main ingredients of the proofs in the **boros research** paper. For the proof to be sound one needs to enforce that only orders within the bounds provided `_calcRateBound` can be matched at all times.

The check performed in this context in `_checkMargin` only guarantees that the new set of orders getting added to the corresponding order books have their rates in the bounds provided (these bounds are dependant on the state of the market and timestamp and thus can change). Let's call the set of bad open orders of user u which fall outside of the required bound $O_{b,u}(t, S)$, where t is the timestamp and S is the state of the market.

There is a functionality that allows an authorised user to call market hut to purge out of bound orders for a set of markets. and thus potentially guarantee that for all users at that timestamp t_p, $O_{b,u}(t_{p+}, S) = \emptyset$ (here t_{p+} represent timestamp t_p after the purge call). Then when a user calls `orderAndOtc` in the same timestamp one could also deduce that $O_{b,u}(t_{p+}, S) = \emptyset$ because of the check in `_checkMargin`.

But time could pass and just before the next purge call the set $O_{b,u}(t_{p+}, S)$ could start having some elements. at this point if someone calls `orderAndOtc` to match against those out of bound bad orders since the bound check is not performed in the matching route, those out of bound orders could get matched and thus breaking one of the invariant assumptions of the paper.

Recommendation: To fully guarantee the invariant assumption in Propostion 5.2 of the boros research paper, one should perform the bound checks for the rates during order matching.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.2.7 Matching amounts

Severity: Low Risk

Context: ProcessMergeUtils.sol#L47-L53

Description/Recommendation: Due to how rounding directions behave it would make more sense for certain quantities that would need to match on both sides and cancel each other to just use the negation of one calculation. For this context for example, it would be best to not use the opposite trade and instead perform:

```
user.signedSize += trade.signedSize();
counter.signedSize -= trade.signedSize();

int256 paymentUser = -trade.toUpfrontFixedCost(market.timeToMat);
int256 paymentCounter = -paymentUser;
```

This would guarantee the following invariant for two mirrored parameters:

$$X + X_{op} = 0$$

Pendle Finance: Acknowledged though we won't fix. Rounding error of 1 wei (18 decimals) is too small.

Spearbit: Acknowledged.

Footnote: The above suggestion applies to many other places where:

- User and counter.
- Winner of force deleverage and the loser.
- ...

8.2.8 Checks missing to make sure immutable parameters of the `_marketHubRiskManagement` match with `MarketHubEntry`

Severity: Low Risk

Context: MarketHubEntry.sol#L35-L45

Description: Checks missing to make sure immutable parameters of the `_marketHubRiskManagement` match with `MarketHubEntry`:

- `_permissionController`.
- `_marketFactory`.
- `_router`.
- `_treasury`.
- `_maxEnteredMarkets`.

Recommendation: Make sure to add a check for the above immutable parameters in the constructor to make sure they match with the ones in `_marketHubRiskManagement`.

Pendle Finance: Won't fix. We'll make sure they get deployed at the same time with same parameters.

Spearbit: Acknowledged.

8.2.9 No lower bound checks in `setRiskyThresHR`

Severity: Low Risk

Context: Storage.sol#L233-L242

Description: `riskyThresHR` which is used to force cancel user orders when the health ratio goes below this value corresponds to H_c in the boros research paper used by the cancel bot B_c. `critHR` seems to correspond to H_f from the boros research paper. Based on the communication with the client during the boros research paper review (Finding 5) we know that we should have:

$$H_f < H_d - D(H_d, 30s) < H_d < H_c - D(H_c, 30s) < H_c$$

Currently the value for H_d is not set in the contracts nor the function $D(H, t)$ is defined, but in general based on above we should have:

$$H_f < H_c - D(H_c, 30s) - D(H_c - D(H_c, 30s), 30s) < H_c$$

Recommendation: It would be best to check in `setRiskyThresHR`:

$$H_f < H_c - D(H_c, 30s) - D(H_c - D(H_c, 30s), 30s)$$

which would require defining the function D or at least:

$$H_f < H_c$$

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

FootNote: Test suite (BorosTestBase.sol#L130-L132):

```
int256 critHR = 0.4e18;      // H_f, at some other places set to 1e18
int256 delevHR = 0.6e18;     // H_d, not used in the test suite
int256 cancelHR = 0.8e18;    // H_c, not used in the test suite
```

margin.t.sol#L971:

```
int256 riskyThresHR = 1.2e18;
```

8.2.10 Incorrect rounding directions

Severity: Low Risk

Context: PMath.sol#L66-L76, PMath.sol#L89-L99

Description: The issue from the first review still applies: First review - Finding 2

Recommendation: Rounding direction needs to be corrected based on the signs of the numerator and denominator.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.3 Gas Optimization

8.3.1 Only read `tThresh` when needed

Severity: Gas Optimization

Context: CoreStateUtils.sol#L230-L238

Description: If the market is matured one does not need to read `tThresh` from storage since it will be set to `0` in memory.

Recommendation: Only read `tThresh` from storage for non-matured markets:

```
function _readMarketExceptMarkRate(MarketMem memory market) internal view {
    market.OI = market.origOI = uint256(_ctx().OI).Int();
    market.status = _ctx().status;
    market.k_maturity = _ctx().k_maturity;
    market.k_tickStep = _ctx().k_tickStep;
    market.k_iThresh = uint128(TickMath.getRateAtTick(int16(_ctx().k_iTickThresh), market.k_tickStep));

    market.latestFTag = _ctx().latestFTag;
    market.latestFTime = _ctx().latestFTime;
    bool isMatured = _isMatured(market);

    if (isMatured) {
        market.timeToMat = 0;
        market.tThresh = 0;
    } else {
        market.timeToMat = market.k_maturity - market.latestFTime;
        market.tThresh = _ctx().tThresh; // <-- this line is moved here
    }
}
```

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.3.2 `batchSimulate` can be optimised

Severity: Gas Optimization

Context: MiscModule.sol#L63-L79

Description: Currently `batchSimulate` if all checks passed decoded the returned data then the compiler when calculating the `return` statement encode the data back to the same structure. One can avoid this unnecessary decoding and encoding which expands the memory more than necessary as well as all the gas costs associated with the ABI decode and encode. Moreover the check for the length of the `result` can be more strict as even if `batchRevert` processes `0` calls it would at least return `4 + 3 * 32` bytes (considering even some ABI-encoding tricks to have the head of two empty dynamic arrays point to the same slot).

Some example:

```
cast cd  "Router_RevertedBatchResult(bytes[] memory results, uint256[] memory gas)" "[]" "[]"
```

```
.   offset:                                                                  data description
------------------------------------------------------------------------------------------------------------
0 + 0x0000:                                                                0185bf65 selector
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000040 results[] head/offset
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000060 gas[] head/offset
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000000 results[] length
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000000 gas[] length
```

The above could potentially be compressed to (the `solc` does not use this trick as of right now):

```
.   offset:                                                                  data description
------------------------------------------------------------------------------------------------------------
0 + 0x0000:                                                                0185bf65 selector
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000040 results[] head/offset
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000040 gas[] head/offset
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000000 0 length
```

Thus why the minimum returned data length of `batchRevert` is `4 + 3 * 32`.

```
cast cd  "Router_RevertedBatchResult(bytes[] memory results, uint256[] memory gas)" "[0xaabb, 0xcc]"
↪  "[1,2]"
```

```
.   offset:                                                                  data description
------------------------------------------------------------------------------------------------------------
0 + 0x0000:                                                                0185bf65 selector
4 + 0x0000: 0000000000000000000000000000000000000000000000000000000000000040 results[] head/offset
4 + 0x0020: 0000000000000000000000000000000000000000000000000000000000000120 gas[] head/offset
4 + 0x0040: 0000000000000000000000000000000000000000000000000000000000000002 results[] length
4 + 0x0060: 0000000000000000000000000000000000000000000000000000000000000040 results[0] head
4 + 0x0080: 0000000000000000000000000000000000000000000000000000000000000080 results[1] head
4 + 0x00a0: 0000000000000000000000000000000000000000000000000000000000000002 results[0] length
4 + 0x00c0: aabb000000000000000000000000000000000000000000000000000000000000 results[0] with padding
4 + 0x00e0: 0000000000000000000000000000000000000000000000000000000000000001 results[1] length
4 + 0x0100: cc000000000000000000000000000000000000000000000000000000000000 results[1] with padding
4 + 0x0120: 0000000000000000000000000000000000000000000000000000000000000002 gas[] length
4 + 0x0140: 0000000000000000000000000000000000000000000000000000000000000001 gas[0]
4 + 0x0160: 0000000000000000000000000000000000000000000000000000000000000002 gas[1]
```

Recommendation: To avoid unnecessary decoding and encoding and also enforce stricter length check for the returned data, one can change `batchSimulate` to:

```
function batchSimulate(
    SimulateData[] memory calls
) external returns (bytes[] memory /*results*/, uint256[] memory /*gasUsed*/) {
```

```
    (bool success, bytes memory result) = address(this).delegatecall(abi.encodeCall(this.batchRevert,
    ↪  calls));
    assert(!success);

    if (
        result.length < (4 + 3 * 32) || // stricter length check
        bytes4(result) != Router_RevertedBatchResult.selector
    ) {
        _revertBytes(result);
    }

    // memory is not modified in the block below
    assembly ("memory-safe") {
        return(
            add(result, 0x24), // 32 (length slot) + 4 (custom error selector)
            sub(mload(result),4)
        )
    }
}
```

Please add unit tests to ensure robustness. The above new implementation relies on the fact that if `this.batchRevert` delegate call returns with a `Router_RevertedBatchResult.selector` then the proceeding data are encoded correctly as currently implemented in `batchRevert` to return `(bytes[], uint256[])`.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.4 Informational

8.4.1 `remove` for `MarketId` arrays only removes the first instance found

Severity: Informational

Context: ArrayLib.sol#L15-L24

Description: `remove` for `MarketId` arrays only removes the first instance found and returns `true` early. If there are multiple elements in the array with the same `id` to be removed, only the first one gets removed and the rest remain. Currently due to the call sites for:

- User's entered markets.
- `_strictMarkets`.

One can prove that these arrays would not have duplicate elements.

Recommendation: If the implementation of this function is going to only remove the first found element, this should be documented as a comment or it should be documented that this function assumes the array provided does not have duplicate elements.

Pendle Finance: Fixed in commit e2aa73d0.

Spearbit: NatSpec comment has been added to indicate the above.

8.4.2 Unsafe memory allocation

Severity: Informational

Context: LibOrderIdSort.sol#L51-L55, ArrayLib.sol#L119-L133

Description: In the new codebase the following functions has been introduced:

- `allocOrderIdArrayNoInit`.
- `allocSweptFArrayNoInit`.

- Inlined memory allocation in `LibOrderIdSort.makeTempArray`.

to replace dynamic memory declarations like `<TYPE>[] memory x = new <TYPE>[](len)`. The custom implementations avoid the following actions performed by the compiler:

1. Memory expansion is not checked to make sure it does not overflow or grow more than `0xffffffffffffffff`.
2. The expanded memory is not cleared out (zeroed out) in case there are dirty bits in those regions (highly unlikely unless custom memory managements are used as opposed to how the compiler `solc` manages the memory where the area pointed to by the free memory pointer onwards should have no dirty bits).
3. And for the case of `allocSweptFArrayNoInit` only the memory space for the head of the array elements are created. This adds the optimisation of not expanding memory more than needed for the elements that currently would occupy 2 words of memory (`64` bytes for `SweptF` struct type). And so only the space for the pointers to the elements in the memory are allocated. For an array of size n that would mean $n + 1$ memory slots/words versus $n + 1 + 2n$ as the full array might not get utilised and the length would be set to a smaller number. This would mean that the memory layout for these arrays `SweptF[] memory res` would be fragmented in general whereas in the old implementation this memory layout would have been contiguous.

Recommendation: The above should be documented for the devs and the users and perhaps if not necessary due to gas optimisations it would be best to revert back to the old implementation. If not, it would be best to at least include the check required for **1.** For example the compiler includes this snippet when updating the free memory pointer (allocating memory):

```
function finalize_allocation(memPtr, size) {
    let newFreePtr := add(memPtr, round_up_to_mul_of_32(size))
    // protect against overflow
    if or(gt(newFreePtr, 0xffffffffffffffff), lt(newFreePtr, memPtr)) { panic_error_0x41() }
    mstore(64, newFreePtr)
}
```

For (2) the compiler uses the following trick to zero out chunks of memory:

```
function zero_memory_chunk_rich_id(dataStart, dataSizeInBytes) {
    calldatacopy(dataStart, calldatasize(), dataSizeInBytes)
}
```

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.4.3 Typo in `_settleProcess`

Severity: Informational

Context: MarginManager.sol#L287

Description/Recommendation: Typo in `_settleProcess`'s return parameter list. The last `,` should come before the commented out parameter:

```
(VMResult thisVM, PayFee thisPayFee, int256 thisSignedSize, /*uint256 nOrders*/) =
//                                                          ^^^
```

Pendle Finance: Fixed in commit 23fbc140.

Spearbit: Fix verified.

8.4.4 `entranceFee` is only paid once and is indexed by token ids

Severity: Informational

Context: MarginManager.sol#L150-L155

Description: each user `acc` has a flag per `marketId`, `acc[user].hasEnteredMarketBefore[marketId]`, which ensures that the market entrance fees are only paid once to the treasury. So for example if a user enter into a market then exits and then enters into the same market again, it would not pay the entrance fee the second time.

The entrance fees are indexed based on token ids which can be changed by authorised roles. The question arises that what if the entrance fee was low or even `0` the first time the user entered into the market and later, the risk managers decide that this value is very low and should charge a higher fee. Then in this case if the user exits and re-enters back into the same market it would not pay the fees anymore.

Recommendation:

1. The scenario above should be considered and perhaps enforce the user to pay fees each time enters back into the markets. Otherwise it should be documented for the risk managers.
2. It would be best to have a more granular `marketEntranceFee` indexed by the `marketId`s instead of token ids. Since multiple markets can have the same token ids but could have different risk factors.

Pendle Finance: Acknowledged.

Spearbit: Acknowledged.

8.4.5 `getMarketConfig()` returns the actual storage value for `tThresh`

Severity: Informational

Context: MarketOffView.sol#L154

Description: `getMarketConfig()` returns the actual storage value for `tThresh`. Whereas the used value is the same as `tThresh` for non-matured markets and `0` for matured markets.

Recommendation: Either add a NatSpec to mention that `tThresh` returned by `getMarketConfig()` is the raw value or make sure to return `0` for matured markets.

Pendle Finance: Acknowledged. It's named `Config` so it should be raw value.

Spearbit: Acknowledged.

9 Appendix

9.1 Known Bugs Reported by Pendle Finance

The following Bugs were communicated to the Cantina team by the client:

- MarketTypes.sol#L567-L569: `AccountData2Lib.signedSize(...)` is incorrect but unused and will be deleted:

```
function signedSize(AccountData2 data) internal pure returns (int128) {
    return int128(uint128(AccountData2.unwrap(data)));
}
```

  The value returned should have been:

```
int128(uint128(AccountData2.unwrap(data) >> 64))
```

- Order.sol#L21-L25: `LONG` & `SHORT` are currently constants, which doesn't have the enum built-in range check. Will be changed back to `enum`.
- MarginManager.sol#L141-L188: The margin checks aren't robust enough for the case where the user creates a stop order, because the `marginRate` at which the stop order is filled will be worse than the current mark rate that's used to calculate the margin. When the stop order gets filled, it is possible for the user's health ratio to fall below 1 and become immediately liquidatable.

```
function test_instantLiquidatableWorseRate() public {
    vm.startPrank(admin);
    // set fees to 0
    market0.setFeeRates(0, 0);
    // set mark rate to 10%
    setMarkRate(rateOracle, 10e16);

    // set IM factor to 100%, MM factor to 80%
    IMFactor = 100e16;
    MMFactor = 80e16;
    market0.setMarginConfig(IMFactor, MMFactor);
    vm.stopPrank();


    int256 totalValue = marketHub.accCash(toCrossAcc(alice));
    int256 maxSize = calcMaxSize(totalValue, IMFactor, PMath.ONE, 0);
    console.log("maxSize", maxSize);
    vm.prank(alice);
    router.placeOrderALO(true, market0Id, LONG, uint256(maxSize), 3646); // interest rate of
    ↪  ~20%
    int256 aliceTotalValue = getTotalValue(alice);
    console.log("aliceTotalValue", aliceTotalValue);
    int256 aliceIM = getTotalIM(alice).Int();
    console.log("aliceIM", aliceIM);
    int256 aliceMM = getTotalMM(alice).Int();
    console.log("aliceMM", aliceMM);

    // set mark rate to 20%
    vm.prank(admin);
    setMarkRate(rateOracle, 20e16);
    vm.prank(bob);
    router.placeOrderFOK(true, market0Id, SHORT, uint256(maxSize), type(int16).min);

    // check if liquidatable
    aliceTotalValue = getTotalValue(alice);
    console.log("aliceTotalValueAfter", aliceTotalValue);
    aliceIM = getTotalIM(alice).Int();
```

```
        console.log("aliceIMAfter", aliceIM);
        aliceMM = getTotalMM(alice).Int();
        console.log("aliceMMAfter", aliceMM);
        int256 healthRatio = getHealthRatio(alice);
        console.log("healthRatio", healthRatio);
        assertGt(healthRatio, 0);
        assertLt(healthRatio, PMath.IONE);

        // attempt liquidation, shouldn't revert
        vm.prank(bob);
        router.liquidate(true, marketOId, toCrossAcc(alice), 1 ether);
    }
```