Bumper Finance Audit Report

Aug 30, 2023

Table of Contents

Summary

This report has been prepared for Bumper Finance smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Bumper Finance**
Codebase	**https://github.com/bumper-dao/protocol**
Commit	**5f6cc088d719f87d097b5643912ea2fb92045c94**
Language	**Solidity**

Audit Summary

Delivery Date	**Aug 30, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**23**

[WP-H1] `closeNative()` will lose the bond to the Taker contract.

High

Issue Description

In `closeNative()`, `closeTo()` will be called with `address(this)` i.e., the Taker contract as the `to_` parameter. As a result, the bond and incentive will be transferred to the Taker contract instead of the user.

https://github.com/bumper-dao/protocol/blob/1be7ec8336b6161221cc90475b0ac4a9a0e0c1fa/contracts/Taker.sol#L86-L93

```
86  function closeNative(uint256 positionId_) public virtual {
87      closeTo(positionId_, false, address(this));
88
89      uint256 amount = IERC20Upgradeable(WETH).balanceOf(address(this));
90      IWETH(WETH).withdraw(amount);
91
92      Address.sendValue(payable(msg.sender), amount);
93  }
```

https://github.com/bumper-dao/protocol/blob/1be7ec8336b6161221cc90475b0ac4a9a0e0c1fa/contracts/Taker.sol#L99-L190

```
99  function closeTo(
100     uint256 positionId_,
101     bool isClaim_,
102     address to_
103 ) public virtual {
    @@ 104,180 @@
181
182     deps.bondController.unlockTokensTo(
183         to_,
184         position.bondAmount + position.incentiveAmount
185     );
186
187     delete takerPositions[positionId_];
188
```

```
189         market.rebalanceAndSwap(tempState, diffAsset, diffCapital, true);
190     }
```

Recommendation

Consider adding a new parameter to `closeTo()` called `bondTo_` for the caller to specify the recipient address for the bond, and use `msg.sender` as `bondTo_` in `closeNative()` .

Or:

```
99  function closeTo(
100     uint256 positionId_,
101     bool isClaim_,
102     address to_
103 ) public virtual {
    @@ 104,180 @@
181
182     deps.bondController.unlockTokensTo(
183         to_ == address(this) ? msg.sender : to_,
184         position.bondAmount + position.incentiveAmount
185     );
186
187     delete takerPositions[positionId_];
188
189     market.rebalanceAndSwap(tempState, diffAsset, diffCapital, true);
190 }
```

Status

✓ Fixed

[WP-H2] Reentrancy on `_safeMint()`

High

Issue Description

Attacker can reenter the protocol and change the state of market at L60 of `Maker#depositFor()`, while the dirty state from L54 will continue to be used at L65 and L100.

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/Maker.sol#L44-L116

```
44  function depositFor(
45      address for_,
46      uint256 amount_,
47      uint32 tier_,
48      uint16 termDays_
49  ) public virtual returns (uint256) {
50      if (amount_ <= 0) {
51          revert InvalidAmount();
52      }
53
54      IMarket.UpdatedState memory tempState = market.getUpdatedState(true);
55
56      IConfigurableMarket.Dependencies memory deps = _deps();
57
58      deps.capitalVault.depositFrom(msg.sender, amount_);
59
60      uint256 positionId = _mint(for_);
61
62      MakerPosition memory position;
63      (tempState, position) = deps.model.stateAfterMakerEnter(
64          IModel.StateAfterMakerEnterInputParams({
65              currentState: tempState,
@@ 66,78 @@
79          })
80      );
81
@@ 82,97 @@
```

```
98
99      market.rebalanceAndSwap(
100         tempState,
101         0,
102         int256(position.capitalAmount),
103         false
104     );
105
        @@ 106,115 @@
116 }
```

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/PositionToken.sol#L84-L88

```
84  function _mint(address to_) internal returns (uint256 newTokenId) {
85      newTokenId = _tokenIdCounter.current();
86      _tokenIdCounter.increment();
87      _safeMint(to_, newTokenId);
88  }
```

Recommendation

Consider not using `_safeMint()` :

```
84  function _mint(address to_) internal returns (uint256 newTokenId) {
85      newTokenId = _tokenIdCounter.current();
86      _tokenIdCounter.increment();
87      _mint(to_, newTokenId);
88  }
```

Status

✓ Fixed

[WP-M3] Lack of methods to withdraw the protocol fees.

Medium

Issue Description

Based on the context and how the `assetTreasury` and `capitalTreasury` are maintained, they should be the storage variables that store the amount of pending protocol fees. However, there is no way to withdraw them.

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/model/Model.sol#L578-L580

```
578  p_.currentState.state.debt -= (vars.expiredFee + vars.protocolFee);
579  p_.currentState.state.capitalTreasury += (vars.expiredFee +
580      vars.protocolFee);
```

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/model/Model.sol#L51-L60

```
51  uint256 fee = takerProtocolFee(input_.assetAmount, input_.term);
52
53  if (fee >= input_.assetAmount) {
54      IPositionManager.TakerPosition memory emptyPos;
55
56      return (input_.currentState, emptyPos, fee);
57  }
58
59  uint256 amountMinusFee = input_.assetAmount - fee;
60  input_.currentState.state.assetTreasury += fee;
```

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/Market.sol#L212-L224

```
212  function _swapAndReconcileState(
213      IMarket.UpdatedState memory state_
214  ) internal virtual returns (IMarket.UpdatedState memory) {
```

```
215     (int256 deltaAssets, int256 deltaCapital) = deps
216         .rebalancer
217         .getSwapAmounts(
218             state_.rebalanceState.deltaAssets,
219             deps.assetVault.balance() - state_.state.assetTreasury,
220             deps.capitalVault.balance() - state_.state.capitalTreasury,
221             state_.lastVisitedPrice.price,
222             decimals,
223             protocolConfig
224         );
```

Status

✓ Fixed

[WP-M4] `expiredPenalty` should also be removed from `state.debt` when makers renew their position.

Medium

Issue Description

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/model/Model.sol#L487-L589

```
487  function stateAfterMakerRenew(
488      IModel.StateAfterMakerRenewInputParams memory p_
489  )
     @@ 490,498 @@
499  {
     @@ 500,526 @@
527
528      vars.expiredPenalty = makerExpiredPenalty(
529          IModel.MakerExpiredPenaltyInputParams({
530              position: p_.position,
531              atBlockTimestamp: p_.atBlockTimestamp,
532              makerPositiveClaimTokenTotalSupply: p_
533                  .makerPositiveClaimTokenTotalSupply,
534              makerNegativeClaimTokenTotalSupply: p_
535                  .makerNegativeClaimTokenTotalSupply,
536              debt: p_.currentState.state.debt,
537              makerCapital: vars.makerCapital
538          })
539      );
540
     @@ 541,571 @@
572
573      // TODO: @sam to confirm
574      p_.currentState.state.yieldTarget -=
575          (p_.currentState.state.yieldTarget *
576              (vars.expiredFee + vars.protocolFee)) /
577          p_.currentState.state.debt;
578      p_.currentState.state.debt -= (vars.expiredFee + vars.protocolFee);
579      p_.currentState.state.capitalTreasury += (vars.expiredFee +
```

```
            vars.protocolFee);

        return (
            p_.currentState,
            p_.position,
            vars.protocolFee,
            vars.expiredFee,
            vars.expiredPenalty
        );
    }
```

`expiredPenalty` should be removed from the `state.debt` so that it can be socialized among the other stakeholders (in `computeMakerWithdrawalAmount()`).

This is properly handled in maker withdrawal as the entire `capitalAmount`, which includes `expiredPenalty`, is removed from the `state.debt`:

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/model/Model.sol#L436-L482

```
    uint256 capitalShare = computeMakerWithdrawalAmount(
        IModel.ComputeMakerWithdrawalAmountInputParams({
            makerCapitalDeposit: p_.position.capitalAmount,
            makerPositiveClaimAmount: p_.position.positiveClaimAmount,
            makerNegativeClaimAmount: p_.position.negativeClaimAmount,
            makerPositiveClaimTokenTotalSupply: p_
                .makerPositiveClaimTokenTotalSupply,
            makerNegativeClaimTokenTotalSupply: p_
                .makerNegativeClaimTokenTotalSupply,
            debt: p_.currentState.state.debt,
            makerCapital: makerCapital
        })
    );

    uint256 expiredPenalty = makerExpiredPenalty(
        IModel.MakerExpiredPenaltyInputParams({
            position: p_.position,
            atBlockTimestamp: p_.atBlockTimestamp,
            makerPositiveClaimTokenTotalSupply: p_
                .makerPositiveClaimTokenTotalSupply,
```

```
456             makerNegativeClaimTokenTotalSupply: p_
457                 .makerNegativeClaimTokenTotalSupply,
458             debt: p_.currentState.state.debt,
459             makerCapital: makerCapital
460         })
461     );
462
463     p_.currentState.weightedState.positiveRwCapital -= p_
464         .position
465         .positiveRiskWeightedCapital;
466
467     p_.currentState.weightedState.negativeRwCapital -= p_
468         .position
469         .negativeRiskWeightedCapital;
470
471     p_.currentState.state.yieldTarget -=
472         (p_.currentState.state.yieldTarget *
473             (p_.position.capitalAmount + expiredFee)) /
474         p_.currentState.state.debt;
475
476     p_.currentState.state.debt -= (p_.position.capitalAmount + expiredFee);
477
478     p_.currentState.state.capitalTreasury += expiredFee;
479
480     (capitalShare, expiredPenalty) = capitalShare.safeFeeSub(
481         expiredPenalty
482     );
```

Recommendation

Change to:

```
574     p_.currentState.state.yieldTarget -=
575         (p_.currentState.state.yieldTarget *
576             (vars.expiredFee + vars.protocolFee)) /
577         p_.currentState.state.debt;
578     p_.currentState.state.debt -= (vars.expiredFee + vars.protocolFee +
        vars.expiredPenalty);
579     p_.currentState.state.capitalTreasury += (vars.expiredFee +
580         vars.protocolFee);
```

Status

✓ Fixed

[WP-M5] `maxUpdatePriceIterations` feature sometimes does not work because of a misjudgment in `ChainLinkCombinedFeed.canCatchUp()` .

Medium

Issue Description

Due to a mistake in `ChainLinkCombinedFeed` at line 68, where `endToRoundId` was mistakenly used as `startToRoundId` , this caused the calculation `toRoundId_ - startToRoundId` at line 76 to become ineffective.

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/Rebalancer.sol#L35-L84

```
35  function getUpdatedState(
36      IRebalancer.GetUpdatedStateInputParams memory p_
37  ) public view returns (IMarket.UpdatedState memory updatedState_) {
38      IMarket.UpdatedStateLoopVars memory localVars;
39      updatedState_ = p_.storedState;
40
41      // if p_.lastVisitedPriceId is 0 we are at protocol start
42      updatedState_.prevVisitedPrice = (0 == p_.lastVisitedPriceId)
43          ? p_.priceFeedAssetToCapital.priceLatest()
44          : p_.priceFeedAssetToCapital.priceAt(p_.lastVisitedPriceId);
45
46      updatedState_.lastVisitedPrice = p_
47          .priceFeedAssetToCapital
48          .priceLatest();
49
50      if (
51          updatedState_.prevVisitedPrice.priceId ==
52          updatedState_.lastVisitedPrice.priceId
53      ) {
54          // we're up to date
55          (updatedState_.prevVisitedPrice, ) = p_
56              .priceFeedAssetToCapital
57              .prevPrice(updatedState_.lastVisitedPrice.priceId);
58
59          (, , updatedState_.beta) = p_.model.computePrfVrfBeta(
```

```
60              IModel.ComputePrfVrfBetaInputParams({
61                  prevPriceItem: updatedState_.prevVisitedPrice,
62                  priceItem: updatedState_.lastVisitedPrice,
63                  awAvgFloorPrice: updatedState_
64                      .weightedState
65                      .averageFloorPrice(),
66                  awAvgMaturity: updatedState_.weightedState.averageMaturity()
67              })
68          );
69
70          return updatedState_;
71      }
72
73      if (
74          p_.inAction &&
75          !p_.priceFeedAssetToCapital.canCatchUp(
76              p_.maxUpdatePriceIterations,
77              updatedState_.prevVisitedPrice.priceId,
78              updatedState_.lastVisitedPrice.priceId
79          )
80      ) {
81          // in a user action, if we need to catch up to more than
    maxUpdatePriceIterations prices,
82          // we abort the action since the price feed is too far behind
83          revert CantCatchUp();
84      }
```

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/pricefeed/ChainLinkCombinedFeed.sol#L41-L83

```
41  function canCatchUp(
42      uint80 iterations_,
43      uint80 fromRoundId_,
44      uint80 toRoundId_
45  ) public view returns (bool) {
46      require(
47          fromRoundId_ <= toRoundId_,
48          "ChainLinkCombinedFeed: rounds reversed"
49      );
50      require(iterations_ > 0, "ChainLinkCombinedFeed: 0 iterations");
51
```

```
52      uint16 fromPhaseId = uint16(fromRoundId_ >> 64);
53      uint16 toPhaseId = uint16(toRoundId_ >> 64);
54
55      if (fromPhaseId == toPhaseId) {
56          return (iterations_ >= (toRoundId_ - fromRoundId_));
57      }
58
59      if ((fromPhaseId + 1) != toPhaseId) {
60          return false;
61      }
62
63      (, uint80 endFromRoundId) = feedRegistry.getPhaseRange(
64          feeds[0].base,
65          feeds[0].quote,
66          fromPhaseId
67      );
68
69      (, uint80 startToRoundId) = feedRegistry.getPhaseRange(
70          feeds[0].base,
71          feeds[0].quote,
72          toPhaseId
73      );
74
75      if (
76          iterations_ >=
77          (endFromRoundId - fromRoundId_) + (toRoundId_ - startToRoundId)
78      ) {
79          return true;
80      }
81
82      return false;
83  }
```

```
110   function getPhaseRange(
111     address base,
112     address quote,
113     uint16 phaseId
114   ) external view returns (uint80 startingRoundId, uint80 endingRoundId);
```

Details of FeedRegistry.getPhaseRange()

contracts/FeedRegistry.sol

```
467     /**
468      * @notice returns the range of proxy round ids of a phase
469      * @param base base asset address
470      * @param quote quote asset address
471      * @param phaseId phase id
472      * @return startingRoundId
473      * @return endingRoundId
474      */
475     function getPhaseRange(
476       address base,
477       address quote,
478       uint16 phaseId
479     )
480       external
481       view
482       override
483       returns (
484         uint80 startingRoundId,
485         uint80 endingRoundId
486       )
487     {
488       Phase memory phase = _getPhase(base, quote, phaseId);
489       require(_phaseExists(phase), "Phase does not exist");
490
491       uint16 currentPhaseId = s_currentPhaseId[base][quote];
492       if (phaseId == currentPhaseId) return _getLatestRoundRange(base, quote,
    currentPhaseId);
493       return _getPhaseRange(base, quote, phaseId);
494     }
```

contracts/FeedRegistry.sol

```
912     function _getLatestRoundRange(
913       address base,
914       address quote,
915       uint16 currentPhaseId
916     )
```

```
917       internal
918       view
919       returns (
920         uint80 startingRoundId,
921         uint80 endingRoundId
922       )
923     {
924       Phase memory phase = s_phases[base][quote][currentPhaseId];
925       return (
926         _getStartingRoundId(currentPhaseId, phase),
927         _getLatestRoundId(base, quote, currentPhaseId)
928       );
929     }
```

contracts/FeedRegistry.sol

```
957     function _getLatestRoundId(
958       address base,
959       address quote,
960       uint16 phaseId
961     )
962       internal
963       view
964       returns (
965         uint80 startingRoundId
966       )
967     {
968       AggregatorV2V3Interface currentPhaseAggregator = _getFeed(base, quote);
969       uint80 latestAggregatorRoundId =
    _getLatestAggregatorRoundId(currentPhaseAggregator);
970       return _addPhase(phaseId, uint64(latestAggregatorRoundId));
971     }
972
973     function _getLatestAggregatorRoundId(
974       AggregatorV2V3Interface aggregator
975     )
976       internal
977       view
978       returns (
979         uint80 roundId
```

```
980       )
981     {
982       if (address(aggregator) == address(0)) return uint80(0);
983       return uint80(aggregator.latestRound());
984     }
```

contracts/FeedRegistry.sol

```
263     /**
264      * @notice get the latest completed round where the answer was updated
265      * @param base base asset address
266      * @param quote quote asset address
267      * @dev overridden function to add the checkPairAccess() modifier
268      *
269      * @notice We advise to use latestRoundData() instead because it returns more
    in-depth information.
270      * @dev Use latestRoundData instead. This does not error if no
271      * answer has been reached, it will simply return 0. Either wait to point to
272      * an already answered Aggregator or use the recommended latestRoundData
273      * instead which includes better verification information.
274      */
275     function latestRound(
276       address base,
277       address quote
278     )
279       external
280       view
281       override
282       checkPairAccess()
283       returns (
284         uint256 roundId
285       )
286     {
287       uint16 currentPhaseId = s_currentPhaseId[base][quote];
288       AggregatorV2V3Interface aggregator = _getFeed(base, quote);
289       require(address(aggregator) != address(0), "Feed not found");
290       return _addPhase(currentPhaseId, uint64(aggregator.latestRound()));
291     }
```

Recommendation

Change to:

```
63  (
64      // startFromRoundId
65      , uint80 endFromRoundId
66  ) = feedRegistry.getPhaseRange(
67      feeds[0].base,
68      feeds[0].quote,
69      fromPhaseId
70  );
71
72  (
73      uint80 startToRoundId,
74      // endToRoundId
75  ) = feedRegistry.getPhaseRange(
76      feeds[0].base,
77      feeds[0].quote,
78      toPhaseId
79  );
```

Status

✓ Fixed

[WP-M6] `ChainLinkCombinedFeed#_priceAtRoundId()` can produce extremely inaccurate prices when `feeds.length > 1` .

Medium

Issue Description

`ChainLinkCombinedFeed` is designed to produce a combined price feed from chained feeds. For example, it combines BTC/ETH and ETH/USDC to create BTC/USDC.

`ChainLinkCombinedFeed#_priceAtRoundId()` is supposed to return the historical price at the specific `roundId` , however, when `feeds.length > 1` , all the other feeds besides the base feed will just take the current price.

This can result in extremely inaccurate price for the given `roundId` .

For example, let's say the historical ETH/BTC price was 1 BTC <> 10 ETH at `roundId 1` , and the actual historical ETH price then was 2k USDC per ETH.

The latest ETH price rose to 10k USDC per ETH (the actual BTC price remains unchanged, current ETH/BTC rate is 1 BTC <> 1 ETH, i.e., ETH flippening BTC).

`_priceAtRoundId()` will return a price of 100k USDC per BTC for `roundId 1` , that's 10x more than the actual price at `roundId 1` .

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/pricefeed/ChainLinkCombinedFeed.sol#L162-L214

```
162  function _priceAtRoundId(
163      uint80 roundId_
164  ) internal view returns (IPriceFeed.Item memory, bool) {
165      (
166          uint80 baseFeedRoundId,
167          int256 baseFeedAnswer,
168          ,
169          uint256 baseFeedUpdatedAt,
170
171      ) = feedRegistry.getRoundData(feeds[0].base, feeds[0].quote, roundId_);
172
173      // return early if the round does not exist on the base feed
```

```
174     if (baseFeedUpdatedAt == 0) {
175         return (
176             IPriceFeed.Item({priceId: 0, price: 0, updatedAt: 0}),
177             false
178         );
179     }
180 
181     uint256 computedPrice = _processAggregatorAnswer(
182         baseFeedAnswer,
183         feeds[0].reversed,
184         feedsDecimals[0],
185         18
186     );
187 
188     for (uint256 i = 1; i < feeds.length; i++) {
189         FeedConfig memory feedConfig = feeds[i];
190 
191         (, int256 _answer, , , ) = feedRegistry.latestRoundData(
192             feeds[i].base,
193             feeds[i].quote
194         );
195 
196         uint256 price = _processAggregatorAnswer(
197             _answer,
198             feedConfig.reversed,
199             feedsDecimals[i],
200             18
201         );
202 
203         computedPrice = (computedPrice * price) / 10 ** 18;
204     }
205 
206     return (
207         IPriceFeed.Item({
208             priceId: baseFeedRoundId,
209             price: computedPrice,
210             updatedAt: baseFeedUpdatedAt
211         }),
212         true
213     );
214 }
```

Recommendation

Because the nature of Chainlink's roundId across different feeds cannot be aligned, we recommend that you remove the ability to combine feeds.

Status

✓ Fixed

[WP-H7] Liquidation of taker positions will most certainly be late, `book` will be updated to an incorrect value.

High

Issue Description

In the current implementation, a taker position can only be liquidated when ”it does not have enough assets to cover the premium, penalty, and protocol fee”.

The penalty and protocol fee only apply when the position expires. Therefore, most times, a taker position will get liquidated because they cannot pay for the premium.

Premium will be settled at the global level (`globalPremiumDelta`) first and later applies to the individual taker position by tracking the delta to `takerPremiumCI` .

The global premium will be settled at each price update, and sometimes in batch (`maxUpdatePriceIterations`).

Chainlink’s feed is updated each time the price change surpasses a predefined `deviation` or a certain period of time has passed since the last update (`heartbeat`). In other words, the price update won’t be very gradual.

So that one update or especially a batch update can cause a significant amount of premium to be added to the global level.

That is to say, the premium update that makes a taker position liquidatable can cause a significant amount of shortfall.

When the position becomes liquidatable, that’s already too late to liquidate it, as it’s already causing bad debt to the protocol.

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/model/Model.sol#L133-L214

```
133  function stateAfterTakerClose(
134      StateAfterTakerCloseInputParams memory input_,
135      bool isLiquidation_
136  )
```

```
137     public
138     view
139     returns (IMarket.UpdatedState memory, uint256, uint256, bool)
140 {
        @@ 141,159 @@
160     if (input_.position.assetAmount <= amountDue) {
161         if (!isLiquidation_) {
162             // position does not have enough assets to cover premium, penalty and
        protocol fee
163             // -> position must be liquidated
164             return (input_.currentState, 0, expiredProtocolFee, true);
165         } else {
166             amountOut = 0;
167             expiredProtocolFee = 0;
168             // expiredPenalty does not matter because it is not used anywhere
169
170             if (input_.position.assetAmount > totalPremium) {
171                 // premium was already deducted from book by update loop
172                 uint256 assetsRemaining = input_.position.assetAmount -
173                     totalPremium;
174
175                 // per spec: expiredPenalty is paid first, then protocol fee if
        anything left
176                 if (assetsRemaining > expiredPenalty) {
177                     expiredProtocolFee = assetsRemaining - expiredPenalty;
178                 }
179             }
180         }
181     } else {
182         if (isLiquidation_) {
183             // shouldn't be liquidation because position has enough assets to cover
        premium, penalty and protocol fee
184             return (input_.currentState, 0, expiredProtocolFee, false);
185         }
186
187         amountOut = input_.position.assetAmount - amountDue;
188     }
189
190     input_.currentState.state.assetTreasury += expiredProtocolFee;
191
192     input_.currentState.state.book += totalPremium;
193     input_.currentState.state.book -= input_.position.assetAmount;
```

```
194
      @@ 195,213 @@
214   }
```

PoC

1. Alice opened a taker position with 20.1 ETH for 30 days.
2. Due to the market situation, the premium rate has been high. Each day, 1 ETH will be charged from the taker position.
3. On day 21, there is only 0.1 ETH left. After the global premium update, 1 ETH is to be deducted from the position, which makes it liquidatable.
4. In `stateAfterTakerClose()` L192, `book` will be wrongfully increased by 1 ETH that the taker position cannot and did not pay for.

Recommendation

To maintain the integrity of `book`, it is not acceptable for the total premium amount to exceed the `position.assetAmount`. A buffer should be added to allow for liquidation of the position before it becomes insufficient to pay the premium.

Bumper's Response

The project will operate a liquidation bot.

The Book is made whole in instances where liquidation does not occur in a timely fashion, i.e. always restored to the state before the Taker joined (otherwise the Book would not represent the remaining Takers anymore). In other words, prior PaP calculations where the Taker position was greater than 100% accumulated premium, would have overcharged the remaining Takers. This is accounted for in the Book being decreased. By restoring the Book, remaining Taker positions are restored, and rebalancing redistributes the remaining liquidity.

Fee is the liquidation threshold, but this is only present for expired positions.

For a position still within term, there is no liquidation threshold. This is a situation that should not occur in the near term, and a liquidation threshold is intended to be implemented in a future update.

Status

ⓘ Acknowledged

[WP-H8] Lack of incentive for the liquidator to liquidate small positions.

High

Issue Description

The liquidator is incentivized to liquidate a position as they can impound the bond.

However, because the bond is a proportion (2.5%) of the position value. For a small enough position, the value of the bond won't be enough to cover the gas cost to liquidate the position.

Additionally, the bond token price may drop and there is a design that renewal doesn't require additional bond even when the bond value decreases.

With all these combined, for small enough positions, it would be economically disincentivized for the liquidator to take action even when they are very liquidatable.

Liquidation is critical to the protocol. For instance, the payment of premiums is guaranteed by liquidation of taker positions.

If a taker position cannot get liquidated in time, they will continuously create fake premiums which they cannot pay and did not pay to the makers.

The attacker can exploit this by creating a lot of small taker positions, farming inflated premiums by taking a large maker position, and harvesting the premium that they created from these small maker positions.

As a side effect, the book value will be significantly impacted by the fake premiums, which further impacts maker capital.

https://github.com/bumper-dao/protocol/blob/8e32e0de79096f4d42a162f55d7a88d7ff253c81/contracts/Rebalancer.sol#L151-L167

```
151       // because premiumPerShareDelta incurs some precision loss,
152       // premiumDelta is recomputed to account for some of the loss when subtracted
      from the book
```

```
153         updatedState_.state.book += localVars.premiumDeltaPerRun;
154
155         uint256 premiumPerShareDelta;
156         (premiumPerShareDelta, localVars.premiumDeltaPerRun) = p_
157             .model
158             .premiumPerShareDelta(
159                 localVars.premiumDeltaPerRun,
160                 updatedState_.weightedState.rwAssets
161             );
162
163         updatedState_.state.book -= localVars.premiumDeltaPerRun;
164         updatedState_.weightedState.takerPremiumCI += premiumPerShareDelta;
165
166         return updatedState_;
167     }
```

Recommendation

Consider asking for a fixed amount of liquidation deposit denominated in the native token when opening the position for the first time (similar to GMX, MakerDAO, Liquity), and combine it with the bond as the liquidator's incentive, making sure it's always profitable for the liquidator to liquidate a position.

Status

ⓘ Acknowledged

[WP-H9] Expired positions will wrongly shift `awAvgMaturity` to an earlier time, resulting in a wrong `beta` value.

High

Issue Description

https://github.com/bumper-dao/protocol/blob/f39d6ba867e6b20cbd405d81e01a687e92e7ca9b/contracts/model/Model.sol#L97-L102

```
97  input_.currentState.weightedState.awMaturities +=
98      amountMinusFee *
99      (input_.currentTimestamp + input_.term * 24 hours);
100 input_.currentState.weightedState.awFloorPrices +=
101     amountMinusFee *
102     floorPrice;
```

https://github.com/bumper-dao/protocol/blob/8e32e0de79096f4d42a162f55d7a88d7ff253c81/contracts/libraries/state/WeightedStateLib.sol#L7-L14

```
7  function averageMaturity(
8      IMarket.WeightedState memory weightedState_
9  ) internal pure returns (uint256) {
10     return
11         (weightedState_.assetsDeposited == 0)
12             ? 0
13             : weightedState_.awMaturities / weightedState_.assetsDeposited;
14 }
```

PoC

Let's say there are only 2 taker positions, both having the same size:

The maturity of position 1 is one year ago, and the maturity of position 2 is one year later.

For a matured/expired position, the system considers it as a continuous position, which means the expected `awAvgMaturity` should be 6 months later.

However, in the current implementation, the `awAvgMaturity` would be today.

As a result, the VRF would be wrong and further make the beta value wrong.

Bumper's Response

Expired positions will cause beta to drift upwards towards its upper bound, causing an overall increase in premia. However, positions that are still active in a fixed term have a premium discount relative to expired positions. Thus, the increase in premia should be mostly or wholly borne by expired positions. The premium penalties that are levied against expired positions are intended to incentivize those positions to be closed/claimed or renewed. If it is observed in the production system that the average expired position time is increasing, then an increase to the premium penalty rate should be executed to further disincentivize takers remaining in an expired state.

Status

ⓘ **Acknowledged**

[WP-H10] Unfair PNL distribution among the maker positions

High

Issue Description

New makers are taking a portion of the existing makers' profit and loss, because the PNL of a maker position is calculated at the time they withdraw as a portion of the entire maker pool's PNL, based on the shares of ClaimAmounts they received when they opened the position.

https://github.com/bumper-dao/protocol/blob/f39d6ba867e6b20cbd405d81e01a687e92e7ca9b/contracts/model/Model.sol#L709-L731

```
709  function computeMakerWithdrawalAmount(
710      IModel.ComputeMakerWithdrawalAmountInputParams memory p_
711  ) public pure returns (uint256) {
712      // per spec:
713      // Withdrawal = d + (bUSDj / bUSDTOTAL) * Y - OutstandingFeesj, t  :  Surplus
714      // Withdrawal = d + (bUSDn / bUSDnTOTAL) Y - OutstandingFeesj, t  :  otherwise
715      // prettier-ignore
716
717      if (p_.makerCapital >= p_.debt) {
718          // prettier-ignore
719          return p_.makerCapitalDeposit
720              + (p_.makerCapital - p_.debt) * p_.makerPositiveClaimAmount /
     p_.makerPositiveClaimTokenTotalSupply;
721      }
722
723      // prettier-ignore
724      uint256 loss = (p_.debt - p_.makerCapital) * p_.makerNegativeClaimAmount /
     p_.makerNegativeClaimTokenTotalSupply;
725
726      if (loss >= p_.makerCapitalDeposit) {
727          return 0;
728      }
729
730      return p_.makerCapitalDeposit - loss;
731  }
```

When the maker pool has unrealized PNL (which it always does), the new maker will

automatically receive a portion of the profit or loss by the time they open the position and receive their `positiveClaimAmount` and `negativeClaimAmount`.

In other words, when the maker side as a whole is in profit, any new maker will receive a slice of the profit when they enter.

If we assume the minimum term is 30 seconds, anyone would be able to open and close a new maker position and immediately claim a portion of the previous makers' earnings.

Vice versa, the new maker is forced to bear a portion of the unrealized loss even if they have nothing to do with the previous loss.

PoC

- takerA: opened a taker position for 90 days
- makerA: opened a maker position for 120 days

After 90 days:

- takerA closed the taker position;
- makerB opened a maker position for 30 days;

At this stage, there are no taker positions in the market.

After 30 days:

- makerA and makerB withdraw their positions

While Maker B doesn't take any counterparty risk, they are still able to claim a portion of the entire earnings of the maker pool.

Status

ⓘ Acknowledged

[WP-M11] TakerRenew will revert in `capitalToAsset()` because `priceBumpToUsd` is 0.

Medium

Issue Description

`stateAfterTakerRenew() -> stateAfterTakerEnter() -> computeTakerIncentiveAmount()` will revert because `input_.priceBumpToUsd` is `0`, so `capitalToAsset()` will revert due to division by 0 at L50 or L56 in `PriceLib.sol`.

https://github.com/bumper-dao/protocol/blob/f39d6ba867e6b20cbd405d81e01a687e92e7ca9b/contracts/model/Model.sol#L218-L279

```
218  function stateAfterTakerRenew(
219      StateAfterTakerRenewInputParams memory input_
220  )
221      public
222      view
223      returns (
224          IMarket.UpdatedState memory,
225          IPositionManager.TakerPosition memory,
226          uint256
227      )
228  {
     @@ 229,252 @@
253      (
254          input_.currentState,
255          newPosition,
256          newProtocolFeeAsset
257      ) = stateAfterTakerEnter(
258          StateAfterTakerEnterInputParams({
259              currentState: input_.currentState,
260              assetAmount: assetsRemaining,
261              tier: input_.newTier,
262              term: input_.newTerm,
263              priceBumpToUsd: 0,
264              decimals: input_.decimals,
265              currentTimestamp: input_.currentTimestamp,
266              riskRatingRegistry: input_.riskRatingRegistry
```

```
267             })
268         );
269
270         // per spec: bond amount does not change at renew
271         newPosition.bondAmount = input_.oldPosition.bondAmount;
272         newPosition.incentiveAmount += input_.oldPosition.incentiveAmount;
273
274         return (
275             input_.currentState,
276             newPosition,
277             expiredProtocolFeeAsset + newProtocolFeeAsset
278         );
279     }
```

https://github.com/bumper-dao/protocol/blob/f39d6ba867e6b20cbd405d81e01a687e92e7ca9b/contracts/model/Bonding.sol#L49-L122

```
49  function computeTakerIncentiveAmount(
50      ComputeTakerIncentiveAmountInputParams memory p_
51  ) public view virtual returns (uint128) {
    @@ 52,64 @@
65
66      uint256 firstTerm = ((depositInCapital *
67          p_.termDays *
68          THETA_TAKER_COORD) / PrecisionLib.PERCENTAGE_PRECISION)
69          .capitalToAsset(
70              p_.priceBumpToUsd,
71              IMarket.Decimals({
72                  asset: 18, // BUMP has 18 decimals
73                  price: 18, // price oracle has 18 decimals
74                  capital: p_.decimals.capital
75              })
76          );
77
    @@ 78,121 @@
122 }
```

https://github.com/bumper-dao/protocol/blob/f39d6ba867e6b20cbd405d81e01a687e92e7ca9b/contracts/libraries/PriceLib.sol#L34-L57

```
34  function capitalToAsset(
35      uint256 capitalAmount_,
36      uint256 priceAssetToCapital_,
37      IMarket.Decimals memory decimals_
38  ) internal pure returns (uint256 assetAmount_) {
39      if (capitalAmount_ == 0) {
40          return 0;
41      }
42
43      int8 decimalsAdjustment = int8(decimals_.asset) +
44          int8(decimals_.price) -
45          int8(decimals_.capital);
46
47      if (decimalsAdjustment < 0) {
48          return
49              capitalAmount_ /
50              priceAssetToCapital_ /
51              (10 ** uint256(uint8(-decimalsAdjustment)));
52      }
53
54      return
55          (capitalAmount_ * (10 ** uint256(uint8(decimalsAdjustment)))) /
56          priceAssetToCapital_;
57  }
```

Status

✓ Fixed

[WP-H12] The premium will continuously erode the asset in a taker position, but the premium will always be charged based on the original amount.

High

Issue Description

The premium charged on an individual level is based on the original size of the position:

https://github.com/bumper-dao/protocol/blob/8e32e0de79096f4d42a162f55d7a88d7ff253c81/contracts/model/Premium.sol#L431-L455

```
431  function takerAssetPremium(
432      IPositionManager.TakerPosition memory position_,
433      IMarket.WeightedState memory weightedState_
434  ) public view virtual returns (uint256) {
435      if (weightedState_.rwAssets == 0) {
436          return 0;
437      }
438
439      if (position_.assetAmount == 0) {
440          return 0;
441      }
442
443      uint256 commonPremium;
444      unchecked {
445          commonPremium =
446              weightedState_.takerPremiumCI -
447              position_.takerPremiumCIAtStart;
448      }
449
450      // total base premium to be paid for the full position
451      // prettier-ignore
452      return (commonPremium * position_.assetAmount * position_.riskRating)
453          / PrecisionLib.PRECISION
454          / PrecisionLib.PERCENTAGE_PRECISION;
455  }
```

The premium charged on a global level is based on the real-time `book` value:

https://github.com/bumper-dao/protocol/blob/8e32e0de79096f4d42a162f55d7a88d7ff253c81/contracts/model/Premium.sol#L52-L98

```
52  function globalPremiumDelta(
53      IModel.ComputePremiumInputParams memory stateInput_
54  ) public view returns (uint256, uint256) {
55      // per sim:
56      // pap = (self.PRF * lambda_PRF + self.LRF * lambda_LRF) * self.Book_ETH /
    temp_lambda_Premium
57      // temp_lambda_Premium - not used in contracts
58
    @@ 59,88 @@
89
90      // prettier-ignore
91      return (
92          (prf * LAMBDA_PRF + lrf * LAMBDA_LRF)
93          * stateInput_.book
94          / PrecisionLib.PERCENTAGE_PRECISION
95          / PrecisionLib.PERCENTAGE_PRECISION,
96          beta
97      );
98  }
```

PoC

1. Alice opened a taker position with 100 ETH for 365 days.
2. Due to the market situation, the premium rate has been high. Each day, 1 ETH will be charged from Alice's position.
3. 90 days later, there is only 10 ETH left in Alice's position. The `book` value is now 10 ETH.
4. Bob opened another taker position with 10 ETH for 365 days, and the `book` is updated to 20 ETH.

At the next settlement, the global premium is correctly calculated based on the updated `book` value: 20 ETH, but shared among a total of 110 ETH shares.

While Alice and Bob are creating the same risk and receiving the same "service" from the makers, the amount of service fee (`premium`) they paid will be dramatically different:

Alice will be paying for 10/11 of the global premium, and Bob will only be paying 1/11. This is because the proportion of their shares is based on the initial position size rather than the real-time size.

Impact

Old positions (with a higher percentage of their position's size being worn down because of the premium) will be charged for more premium than they should have.

Status

✓ **Fixed**

[WP-H13] When closing a taker position, the system will send funds to the user (`withdrawTo()`) before rebalancing (`rebalanceAndSwap()`). If the assets are insufficient, the withdrawal will revert.

High

Issue Description

https://github.com/bumper-dao/protocol/blob/0128427e61679358069fe0c48d155418f6556cb9/contracts/Taker.sol#L160-L253

```
160  function closeTo(
161      uint256 positionId_,
162      bool isClaim_,
163      address to_,
164      address bondTo_
165  ) public virtual {
     @@ 166,212 @@
213
214      if (isClaim_) {
     @@ 215,229 @@
230      } else {
231          deps.assetVault.withdrawTo(to_, assetOut);
232          diffAsset -= int256(assetOut);
233
234          emit UserAction(
235              msg.sender,
236              to_,
237              positionId_,
238              assetOut,
239              expiredProtocolFeeAsset,
240              UserActionType.TakerClose
241          );
242      }
243
244      deps.bondController.unlockTokensTo(
245          bondTo_,
```

```
246             position.bondAmount + position.incentiveAmount
247         );
248
249         delete takerPositions[positionId_];
250
251         market.rebalanceAndSwap(tempState, diffAsset, diffCapital, true);
252     }
```

Because part of the asset owned by the takers might have been sold to capital, the cash asset may not be enough when a major taker position closes.

Therefore, the system should rebalance first and then try to send funds to the user.

Impact

Major taker position (a position sized greater than the cash asset) cannot be closed.

Status

ⓘ Acknowledged

[WP-M14] The `expiredPenalty` charged when closing a expired taker position may cause a surge of PPS (price per share) of maker's shares (`ClaimAmounts`).

Medium

Issue Description

Unlike regular premium, which is gradually charged at each price update (and also, pending premium will always get settled first before a new maker position is created), the `expiredPenalty` , which can be considered as a kind of premium paid for the perpetual (matured) taker position, will only be charged when the position is being closed.

For a large enough taker position, the `expiredPenalty` can amount to a significant sum.

As a result, the closure of a large taker position will cause a surge in the maker's profit. This can be exploited by front-running the position close transaction and creating a large maker position.

https://github.com/bumper-dao/protocol/blob/0128427e61679358069fe0c48d155418f6556cb9/contracts/model/Model.sol#L135-L216

```
135  function stateAfterTakerClose(
136      StateAfterTakerCloseInputParams memory input_,
137      bool isLiquidation_
138  )
139      public
140      view
141      returns (IMarket.UpdatedState memory, uint256, uint256, bool)
142  {
     @@ 143,157 @@
158
159      uint256 amountDue = totalPremium + expiredPenalty + expiredProtocolFee;
160
161      uint256 amountOut;
     @@ 162,190 @@
```

```
191
192     input_.currentState.state.assetTreasury += expiredProtocolFee;
193
194     input_.currentState.state.book += totalPremium;
195     input_.currentState.state.book -= input_.position.assetAmount;
196
@@ 197,215 @@
216 }
```

Status

ⓘ Acknowledged

[WP-L15] The permissionless `Market.getUpdatedState()` function can be used to emit the `StateUpdate` event, even if there is no actual `_storeUpdatedState()`, which can mess up off-chain applications that consume such events.

Low

Issue Description

https://github.com/bumper-dao/protocol/blob/0128427e61679358069fe0c48d155418f6556cb9/contracts/Market.sol#L151-L190

```
151     /**
152      * @dev update market state prior to user action
153      * @param inAction_ if true, either returns updated state to the latest price from the oracle or revert
154      */
155     function getUpdatedState(
156         bool inAction_
157     ) public virtual returns (IMarket.UpdatedState memory newState_) {
158         IMarket.UpdatedState memory storedState;
159         uint80 _lastVisitedPriceId;
160         // prevVisitedPrice will be computed in getUpdatedState based on lastVisitedPriceId
161         // lastVisitedPrice will be computed in getUpdatedState
162         // beta will be computed in getUpdatedState
163         (
164             storedState.state,
165             storedState.weightedState,
166             storedState.rebalanceState,
167             _lastVisitedPriceId
168         ) = getStoredState();
169 
170         newState_ = deps.rebalancer.getUpdatedState(
171             IRebalancer.GetUpdatedStateInputParams({
172                 storedState: storedState,
173                 model: deps.model,
174                 priceFeedAssetToCapital: deps.priceFeedAssetToCapital,
175                 decimals: decimals,
```

```
176                 lastVisitedPriceId: _lastVisitedPriceId,
177                 maxUpdatePriceIterations: getProtocolConfig()
178                     .maxUpdatePriceIterations,
179                 inAction: inAction_
180             })
181         );
182
183         emit StateUpdate(
184             (state.book - newState_.state.book),
185             (newState_.state.yieldTarget - state.yieldTarget)
186         );
187
188         return (newState_);
189     }
190
```

Recommendation

Consider changing it to a permissioned function:

- Use the internal function `_getUpdatedState()` in `Market` to bypass the permission check;
- Use the authenticated `Market.getUpdatedState()` function in `Taker` and `Maker` .

Status

✓ Fixed

[WP-H16] The case of insufficient maker liquidity to cover the taker's claim

High

Issue Description

When the taker position drops below the `floorPrice`, it relies on the adequacy of maker liquidity at that time in order for the taker to make a claim.

If there is not enough maker liquidity, the maker position cannot make a claim or close. That is, the position is frozen until the maker liquidity recovers. However, given that the maker side is already experiencing a shortfall, anyone who enters the maker side will immediately lose part of their investments. It is economically disadvantageous for new makers.

Also, the fact that the taker must wait for new maker liquidity in order to claim is by itself an unconventional situation that the taker may not expect.

https://github.com/bumper-dao/protocol/blob/87c675544e9bf39127c5d6c0f37caa21242fc031/contracts/Taker.sol#L215-L229

```
215  uint256 capitalOut = assetOut.assetToCapital(
216      position.floorPrice,
217      market.getDecimals()
218  );
219  deps.capitalVault.withdrawTo(to_, capitalOut);
220  diffCapital = -int256(capitalOut);
221
222  emit UserAction(
223      msg.sender,
224      to_,
225      positionId_,
226      capitalOut,
227      expiredProtocolFeeAsset,
228      UserActionType.TakerClaim
229  );
```

Status

ⓘ Acknowledged

[WP-M17] `ChainLinkCombinedFeed#priceLatest()` should revert when `_priceAtRoundId()` returns an invalid price.

Medium

Issue Description

In `ChainLinkCombinedFeed.sol#L143` , there is a possibility of retrieving a roundId that does not exist. In such cases, the `_priceAtRoundId()` function will return a result indicating that the price is not valid (`ChainLinkCombinedFeed.sol#L171`). Instead of considering the 0 price as valid, `priceLatest()` should revert.

https://github.com/bumper-dao/protocol/blob/953cca3b74a848cc463ed936926b143b0dcdb487/contracts/pricefeed/ChainLinkCombinedFeed.sol#L136-L147

```
136  function priceLatest()
137      public
138      view
139      override
140      returns (IPriceFeed.Item memory)
141  {
142      (IPriceFeed.Item memory price, ) = _priceAtRoundId(
143          uint80(feedRegistry.latestRound(feeds[0].base, feeds[0].quote))
144      );
145
146      return price;
147  }
```

https://github.com/bumper-dao/protocol/blob/953cca3b74a848cc463ed936926b143b0dcdb487/contracts/pricefeed/ChainLinkCombinedFeed.sol#L156-L209

```
156      function _priceAtRoundId(
157          uint80 roundId_
158      ) internal view returns (IPriceFeed.Item memory, bool) {
159          (
160              uint80 baseFeedRoundId,
161              int256 baseFeedAnswer,
162              ,
```

```
163             uint256 baseFeedUpdatedAt,
164
165         ) = feedRegistry.getRoundData(feeds[0].base, feeds[0].quote, roundId_);
166
167         // return early if the round does not exist on the base feed
168         if (baseFeedUpdatedAt == 0) {
169             return (
170                 IPriceFeed.Item({priceId: 0, price: 0, updatedAt: 0}),
171                 false
172             );
173         }
174
175         uint256 computedPrice = _processAggregatorAnswer(
176             baseFeedAnswer,
177             feeds[0].reversed,
178             feedsDecimals[0],
179             18
180         );
181
182         for (uint256 i = 1; i < feeds.length; i++) {
183             FeedConfig memory feedConfig = feeds[i];
184
185             (, int256 _answer, , , ) = feedRegistry.latestRoundData(
186                 feeds[i].base,
187                 feeds[i].quote
188             );
189
190             uint256 price = _processAggregatorAnswer(
191                 _answer,
192                 feedConfig.reversed,
193                 feedsDecimals[i],
194                 18
195             );
196
197             computedPrice = (computedPrice * price) / 10 ** 18;
198         }
199
200         return (
201             IPriceFeed.Item({
202                 priceId: baseFeedRoundId,
203                 price: computedPrice,
204                 updatedAt: baseFeedUpdatedAt
205             }),
```

```
            true
        );
    }
```

https://github.com/bumper-dao/protocol/blob/953cca3b74a848cc463ed936926b143b0dcdb487/contracts/Market.sol#L125-L152

```
    function getStoredState()
        public
        view
        virtual
        returns (
            IMarket.State memory,
            IMarket.WeightedState memory,
            IMarket.RebalanceState memory,
            IPriceFeed.Item memory
        )
    {
        if (lastVisitedPriceId == 0) {
            // this is the first time the protocol is processing a price
            return (
                state,
                weightedState,
                rebalanceState,
                priceFeedAssetToCapital.priceLatest()
            );
        }

        return (
            state,
            weightedState,
            rebalanceState,
            priceFeedAssetToCapital.priceAt(lastVisitedPriceId)
        );
    }
```

https://github.com/bumper-dao/protocol/blob/953cca3b74a848cc463ed936926b143b0dcdb487/contracts/Market.sol#L214-L341

```
214     function _getUpdatedState(
215         bool inAction_
216     ) internal virtual returns (IMarket.UpdatedState memory newState_) {
217         (
218             newState_.state,
219             newState_.weightedState,
220             ,
221             newState_.prevVisitedPrice
222         ) = getStoredState();
223         newState_.lastVisitedPrice = priceFeedAssetToCapital.priceLatest();
224
    @@ 225,340 @@
341     }
```

Recommendation

Change to:

```
136 function priceLatest()
137     public
138     view
139     override
140     returns (IPriceFeed.Item memory)
141 {
142     (IPriceFeed.Item memory price, bool isValid) = _priceAtRoundId(
143         uint80(feedRegistry.latestRound(feeds[0].base, feeds[0].quote))
144     );
145
146     require(isValid,  "ChainLinkCombinedFeed: invalid price");
147
148     return price;
149 }
```

Status

✓ Fixed

[WP-L18] Slippage control based on oracle price allow MEV up to the max `TOLERANCE`

Low

Issue Description

https://github.com/bumper-dao/protocol/blob/87c675544e9bf39127c5d6c0f37caa21242fc031/contracts/Rebalancer.sol#L221-L229

```
221   // minAmountOut = toCapitalWithChainlink(abs(deltaAssets_)) * (1 -
      priceImpactTolerance)
222   uint256 minAmountOut = (uint256(-deltaAssets_).assetToCapital(
223       priceAssetToCapital_,
224       decimals_
225   ) *
226       (PrecisionLib.PERCENTAGE_PRECISION -
227           LAMBDA_REBALANCE_PRICE_IMPACT_TOLERANCE)) /
228       PrecisionLib.PERCENTAGE_PRECISION;
```

`LAMBDA_REBALANCE_PRICE_IMPACT_TOLERANCE` is set to 4%, which is significant enough to create a MEV opportunity for the attacker to extract value from the slippage allowance by sandwiching the rebalance transaction.

However, there is no simple quick fix for this. The dilemma is that if we lower this slippage tolerance, the swap may not be able to be fulfilled due to market situation and price feed deviation.

Status

Acknowledged

[WP-L19] Sophisticated takers can take advantage of Oracle delay and create positions at a higher `floorPrice` .

Low

Issue Description

The current implementation uses the latest Oracle price to calculate the floor price.

However, there can be a deviation between the Chainlink feed price and the market price because of the design of Chainlink's feed. The deviation is 0.05% for ETH/USD on mainnet, 0.5% for BTC/USD on mainnet, and even higher for non-major assets, e.g., 1% for UNI/USD.

Sophisticated takers can maximize the use of deviation and create positions with a higher `floorPrice` , increasing the maker's risk.

For instance, if the last update of BTC's price is `$10,100` and the current market price is `$10,001` , Chainlink is not going to update the price as the price hasn't passed the 1% deviation threshold.

A taker can then create a new position using `$10,100` as the "current price", which leads to a higher `floorPrice` and thus higher risk for the makers.

To avoid this, opening positions should be delayed (calculating `TakerPosition.floorPrice` based on the next round's Oracle price).

https://github.com/bumper-dao/protocol/blob/d3fff8982b021262bfcf7296b7df46b74ea48448/contracts/model/Model.sol#L44-L110

```
44      function stateAfterTakerEnter(
45          StateAfterTakerEnterInputParams memory input_
46      )
47          public
48          view
49          virtual
50          returns (
51              IMarket.State memory,
52              IMarket.WeightedState memory,
53              IMarket.TakerPosition memory,
```

```
54              uint256
55          )
56      {
57          uint256 fee = takerProtocolFee(input_.assetAmount, input_.term);
58
59          if (fee >= input_.assetAmount) {
60              IMarket.TakerPosition memory emptyPos;
61
62              return (input_.currentState, input_.weightedState, emptyPos, fee);
63          }
64
65          uint256 amountMinusFee = input_.assetAmount - fee;
66          input_.currentState.assetTreasury += fee;
67
68          uint256 riskRating = riskRatingRegistry.getTakerRiskRating(
69              input_.tier,
70              input_.term
71          );
72
73          uint256 floorPrice = takerFloorPrice(
74              input_.latestPriceAssetToCapital,
75              input_.tier
76          );
77
78          input_.currentState.book += amountMinusFee;
79
80          input_.weightedState.assetsDeposited += amountMinusFee;
81          input_.weightedState.rwAssets += amountMinusFee * riskRating;
82          input_.weightedState.awMaturities +=
83              amountMinusFee *
84              (input_.currentTimestamp + input_.term * 24 hours);
85          input_.weightedState.awFloorPrices += amountMinusFee * floorPrice;
86
87          uint256 bondAmount = computeTakerBondAmount(
88              amountMinusFee.assetToCapital(
89                  input_.latestPriceAssetToCapital,
90                  input_.decimals
91              ),
92              input_.priceBumpToUsd,
93              input_.decimals
94          );
95
96          return (
```

```
            input_.currentState,
            input_.weightedState,
            IMarket.TakerPosition({
                assetAmount: amountMinusFee,
                riskRating: riskRating,
                floorPrice: floorPrice,
                start: input_.currentTimestamp,
                term: input_.term,
                bondAmount: bondAmount,
                takerPremiumCIAtStart: input_.weightedState.takerPremiumCI
            }),
            fee
        );
    }
```

https://github.com/bumper-dao/protocol/blob/d3fff8982b021262bfcf7296b7df46b74ea48448/contracts/Market.sol#L227-L290

```
    /**
     * @dev Called by takers to protect an asset {amount_}
     * @param for_ address to receive claim tokens and position
     * @param amount_ amount deposited
     * @param tier_ protection tier
     * @param term_ protection duration (days)
     */
    function protectFor(
        address for_,
        uint256 amount_,
        uint32 tier_,
        uint16 term_
    ) public virtual nonReentrant {
        IMarket.UpdatedState memory tempState = _getUpdatedState(true);

        assetVault.depositFrom(msg.sender, amount_);

        uint256 positionId = takerPositionNFT.mint(for_);

        uint256 protocolFee;
        (
            tempState.state,
            tempState.weightedState,
```

```
250             takerPositions[positionId],
251             protocolFee
252         ) = model.stateAfterTakerEnter(
253             IModel.StateAfterTakerEnterInputParams({
254                 currentState: tempState.state,
255                 weightedState: tempState.weightedState,
256                 assetAmount: amount_,
257                 tier: tier_,
258                 term: term_,
259                 latestPriceAssetToCapital: tempState.lastVisitedPrice.price,
260                 priceBumpToUsd: priceFeedBUMPToUSD.priceLatest().price,
261                 decimals: getDecimals(),
262                 currentTimestamp: uint32(block.timestamp)
263             })
264         );
265 
266         // this will revert if the user doesn't have enough tokens to create the
    bond
267         bondController.lockTokensFrom(
268             msg.sender,
269             takerPositions[positionId].bondAmount
270         );
271 
272         tempState = rebalancer.rebalance(
273             tempState,
274             int256(takerPositions[positionId].assetAmount),
275             0,
276             true,
277             true
278         );
279 
280         emit UserAction(
281             msg.sender,
282             for_,
283             positionId,
284             amount_,
285             protocolFee,
286             UserActionType.TakerProtect
287         );
288 
289         _storeUpdatedState(tempState);
290     }
```

https://github.com/bumper-dao/protocol/blob/d3fff8982b021262bfcf7296b7df46b74ea48448/contracts/Market.sol#L187-L225

```
187     /**
188      * @dev update market state prior to user action
189      * @param inAction_ if true, either returns updated state to the latest price
    from the oracle or revert
190      */
191     function _getUpdatedState(
192         bool inAction_
193     ) internal virtual returns (IMarket.UpdatedState memory newState_) {
194         IMarket.UpdatedState memory storedState;
195         uint80 _lastVisitedPriceId;
196         // prevVisitedPrice will be computed in getUpdatedState based on
    lastVisitedPriceId
197         // lastVisitedPrice will be computed in getUpdatedState
198         // beta will be computed in getUpdatedState
199         (
200             storedState.state,
201             storedState.weightedState,
202             storedState.rebalanceState,
203             _lastVisitedPriceId
204         ) = getStoredState();
205 
206         newState_ = rebalancer.getUpdatedState(
207             IRebalancer.GetUpdatedStateInputParams({
208                 storedState: storedState,
209                 model: model,
210                 priceFeedAssetToCapital: IPriceFeed(priceFeedAssetToCapital),
211                 decimals: decimals,
212                 lastVisitedPriceId: _lastVisitedPriceId,
213                 maxUpdatePriceIterations: getProtocolConfig()
214                     .maxUpdatePriceIterations,
215                 inAction: inAction_
216             })
217         );
218 
219         emit StateUpdate(
220             (storedState.state.book - newState_.state.book),
221             (newState_.state.yieldTarget - storedState.state.yieldTarget)
222         );
223 
224         return (newState_);
```

```
225     }
```

https://github.com/bumper-dao/protocol/blob/d3fff8982b021262bfcf7296b7df46b74ea48448/contracts/Rebalancer.sol#L61-L187

```
61      function getUpdatedState(
62          IRebalancer.GetUpdatedStateInputParams memory p_
63      ) public view returns (IMarket.UpdatedState memory updatedState_) {
64          IMarket.UpdatedStateLoopVars memory localVars;
65          updatedState_ = p_.storedState;
66
67          // if p_.lastVisitedPriceId is 0 we are at protocol start
68          updatedState_.prevVisitedPrice = (0 == p_.lastVisitedPriceId)
69              ? p_.priceFeedAssetToCapital.priceLatest()
70              : p_.priceFeedAssetToCapital.priceAt(p_.lastVisitedPriceId);
71
72          updatedState_.lastVisitedPrice = p_
73              .priceFeedAssetToCapital
74              .priceLatest();
75
76          if (
77              updatedState_.prevVisitedPrice.priceId ==
78              updatedState_.lastVisitedPrice.priceId
79          ) {
80              // we're up to date
81              (updatedState_.prevVisitedPrice, ) = p_
82                  .priceFeedAssetToCapital
83                  .prevPrice(updatedState_.lastVisitedPrice.priceId);
84
85              (, , updatedState_.beta) = p_.model.computePrfVrfBeta(
86                  IModel.ComputePrfVrfBetaInputParams({
87                      prevPriceItem: updatedState_.prevVisitedPrice,
88                      priceItem: updatedState_.lastVisitedPrice,
89                      awAvgFloorPrice: updatedState_
90                          .weightedState
91                          .averageFloorPrice(),
92                      awAvgMaturity: updatedState_.weightedState.averageMaturity()
93                  })
94              );
95
96              return updatedState_;
```

```
97          }
98
99          if (
100             p_.inAction &&
101             !p_.priceFeedAssetToCapital.canCatchUp(
102                 p_.maxUpdatePriceIterations,
103                 updatedState_.prevVisitedPrice.priceId,
104                 updatedState_.lastVisitedPrice.priceId
105             )
106         ) {
107             // in a user action, if we need to catch up to more than
    maxUpdatePriceIterations prices,
108             // we abort the action since the price feed is too far behind
109             revert CantCatchUp();
110         }
111
112         updatedState_.lastVisitedPrice = updatedState_.prevVisitedPrice;
113
114         while (localVars.iteration < p_.maxUpdatePriceIterations) {
115             (localVars.nextPrice, localVars.canContinue) = p_
116                 .priceFeedAssetToCapital
117                 .nextPrice(updatedState_.lastVisitedPrice.priceId);
118
119             if (!localVars.canContinue) {
120                 break;
121             }
122
123             localVars.iteration++;
124             updatedState_.prevVisitedPrice = updatedState_.lastVisitedPrice;
125             updatedState_.lastVisitedPrice = localVars.nextPrice;
126
127             (localVars.premiumDelta, updatedState_.beta) = p_
128                 .model
129                 .globalPremiumDelta(
130                     IModel.ComputePremiumInputParams({
131                         prevPriceItem: updatedState_.prevVisitedPrice,
132                         priceItem: updatedState_.lastVisitedPrice,
133                         awAvgFloorPrice: updatedState_
134                             .weightedState
135                             .averageFloorPrice(),
136                         awAvgMaturity: updatedState_
137                             .weightedState
138                             .averageMaturity(),
```

```
139                         assetPool: updatedState_.state.assetPool,
140                         assetReserve: updatedState_.state.assetReserve,
141                         capitalPool: updatedState_.state.capitalPool,
142                         capitalReserve: updatedState_.state.capitalReserve,
143                         // confirmed that liability is book * awAvgFloorPrice for
    the purposes of premium
144                         liability: updatedState_.state.book.assetToCapital(
145                             updatedState_.weightedState.averageFloorPrice(),
146                             p_.decimals
147                         ),
148                         book: updatedState_.state.book,
149                         debt: updatedState_.state.debt,
150                         yieldTarget: updatedState_.state.yieldTarget
151                     })
152                 );
153 
154             // dec book
155             localVars.premiumDeltaPerRun += localVars.premiumDelta;
156             updatedState_.state.book -= localVars.premiumDelta;
157 
158             // inc yield target
159             updatedState_.state.yieldTarget += p_.model.globalYieldTargetDelta(
160                 IModel.ComputeYieldTargetInputParams({
161                     prevPriceItem: updatedState_.prevVisitedPrice,
162                     priceItem: updatedState_.lastVisitedPrice,
163                     debt: updatedState_.state.debt
164                 })
165             );
166 
167             // virtual rebalance:
168             updatedState_ = _rebalanceVirtual(updatedState_, 0, 0, p_.decimals);
169         }
170 
171         // because premiumPerShareDelta incurs some precision loss,
172         // premiumDelta is recomputed to account for some of the loss when
    subtracted from the book
173         updatedState_.state.book += localVars.premiumDeltaPerRun;
174 
175         uint256 premiumPerShareDelta;
176         (premiumPerShareDelta, localVars.premiumDeltaPerRun) = p_
177             .model
178             .premiumPerShareDelta(
179                 localVars.premiumDeltaPerRun,
```

```
180                 updatedState_.weightedState.rwAssets
181             );
182
183         updatedState_.state.book -= localVars.premiumDeltaPerRun;
184         updatedState_.weightedState.takerPremiumCI += premiumPerShareDelta;
185
186         return updatedState_;
187     }
```

Status

ⓘ Acknowledged

[WP-I20] Taker position with a lower `floorPrice` may be paying a higher premium

Informational

Issue Description

https://github.com/bumper-dao/protocol/blob/87c675544e9bf39127c5d6c0f37caa21242fc031/contracts/model/Premium.sol#L431-L472

```
431     function takerAssetPremium(
432         IPositionManager.TakerPosition memory position_,
433         IMarket.WeightedState memory weightedState_
434     ) public view virtual returns (uint256) {
435         // TODO: is this needed?
436         if (weightedState_.rwAssets == 0) {
437             return 0;
438         }
439 
440         if (position_.assetAmount == 0) {
441             return 0;
442         }
443 
444         uint256 commonPremium;
445         unchecked {
446             commonPremium =
447                 weightedState_.takerPremiumCI -
448                 position_.takerPremiumCIAtStart;
449         }
450 
451         // total base premium to be paid for the full position
452         // prettier-ignore
453         return (commonPremium * position_.assetAmount * position_.riskRating)
454             / PrecisionLib.PRECISION
455             / PrecisionLib.PERCENTAGE_PRECISION;
456     }
457 
458     /// @inheritdoc IModel
459     function takerExpiredPenalty(
460         IPositionManager.TakerPosition memory position_,
461         uint256 totalPremium_,
```

```
462             uint256 blockTimestamp_
463         ) public view virtual returns (uint256) {
464             if (!position_.isExpired(blockTimestamp_)) {
465                 return 0;
466             }
467 
468             // prettier-ignore
469             return totalPremium_ * position_.expiredFor(blockTimestamp_) *
    LAMBDA_EXPIRED_RATE_TAKER
470                     / position_.activeFor(blockTimestamp_)
471                     / PrecisionLib.PERCENTAGE_PRECISION;
472         }
```

In the Bumper's system, a taker position can be understood as the combination of a European option that will automatically convert into a perpetual option upon maturity. The premium for the perpetual option will be up to 40% more expensive than the European option at the start.

It may or may not be in the best interest of the taker to renew a matured position. If the current price is lower than the price when the position was created, it is most likely more preferable not to renew it, considering that the new floor price will be lower.

In the case that most (maybe even all) positions are in the perpetual mode (matured/expired), the taker positions that pay different premiums may have the same floor price, or the taker positions with the same floor price may be paying a different premium.

We consider this an unfair premium distribution between the takers.

PoC

1. Alice opened a taker position with a size of 10 ETH when the ETH price was 3000 and the floorPrice was 2400.
2. 1 day later, the ETH price dropped to 2500. Bob opened a taker position with the same size, term, and tier, and the floorPrice was 2000.

While Alice's position has a higher floorPrice, which means the risk of payout is higher for the makers, Alice and Bob's positions are paying the same amount of premium.

Status

ⓘ Acknowledged

[WP-G21] Unnecessary `approve(0)` after swap.

Gas

Issue Description

Because `UniswapV2Router02` will use up all the allowance in `swapExactTokensForTokens()` .

Also, the `router` can be changed to immutable for gas saving, given that it can't/won't be changed.

https://github.com/bumper-dao/protocol/blob/0128427e61679358069fe0c48d155418f6556cb9/contracts/swap/UniswapV2Swapper.sol#L9-L47

```
9   contract UniswapV2Swapper is ISwapper {
10      IUniswapV2Router02 public router;
11
12      constructor(address router_) {
13          router = IUniswapV2Router02(router_);
14      }
15
16      function swap(
17          address tokenIn_,
18          address tokenOut_,
19          uint256 amountIn_,
20          uint256 amountOutMin_,
21          address returnTokenOutTo_
22      ) external override returns (uint256 amountOut) {
23          require(
24              IERC20(tokenIn_).approve(address(router), amountIn_),
25              "UniswapV2Swapper: approve failed"
26          );
27
28          address[] memory path = new address[](2);
29          path[0] = tokenIn_;
30          path[1] = tokenOut_;
31
32          uint256[] memory amounts = router.swapExactTokensForTokens(
33              amountIn_,
34              amountOutMin_,
35              path,
```

```
            returnTokenOutTo_,
            block.timestamp
        );

        require(
            IERC20(tokenIn_).approve(address(router), 0),
            "UniswapV2Swapper: approve failed"
        );

        return amounts[amounts.length - 1];
    }
}
```

Recommendation

Change to:

```
contract UniswapV2Swapper is ISwapper {
    IUniswapV2Router02 public immutable router;

    constructor(address router_) {
        router = IUniswapV2Router02(router_);
    }

    function swap(
        address tokenIn_,
        address tokenOut_,
        uint256 amountIn_,
        uint256 amountOutMin_,
        address returnTokenOutTo_
    ) external override returns (uint256 amountOut) {
        require(
            IERC20(tokenIn_).approve(address(router), amountIn_),
            "UniswapV2Swapper: approve failed"
        );

        address[] memory path = new address[](2);
        path[0] = tokenIn_;
        path[1] = tokenOut_;

        uint256[] memory amounts = router.swapExactTokensForTokens(
```

```
33                amountIn_,
34                amountOutMin_,
35                path,
36                returnTokenOutTo_,
37                block.timestamp
38            );
39
40            return amounts[amounts.length - 1];
41        }
42    }
```

Status

✓ Fixed

[WP-N22] No-op code.

Issue Description

https://github.com/bumper-dao/protocol/blob/44aba0c83645dbb22f7cad8ff481a732b1172b9f/contracts/model/Bonding.sol#L49-L64

```
49  function computeTakerIncentiveAmount(
50      ComputeTakerIncentiveAmountInputParams memory p_
51  ) public view virtual returns (uint128) {
52      this;
53      // TODO: this works only if capital is USD - good enough for v1
54      if (
55          (p_.liability <= THETA_TAKER_MIN_LIABILITY) ||
56          (p_.liability > 5 * THETA_TAKER_INFL_ASSETS)
57      ) {
58          return 0;
59      }
60
61      uint256 depositInCapital = p_.takerAssetDeposit.assetToCapital(
62          p_.latestPriceAssetToCapital,
63          p_.decimals
64      );
```

Status

✓ Fixed

[WP-N23] `uint256` variables won't be negative.

Issue Description

https://github.com/bumper-dao/protocol/blob/12e8ab7b277162b11af2567d16cacd287b43dc88/contracts/BondController.sol#L80-L100

```
80  function lockTokensFrom(
81      address user,
82      uint256 amount,
83      uint256 incentiveAmount
84  ) public onlyBondingManager {
85      if (amount <= 0 && incentiveAmount <= 0) {
86          return;
87      }
88
89      if (balanceOf(user) < amount) {
90          revert InsufficientBondBalance();
91      }
92
93      if (incentivePool < incentiveAmount) {
94          revert InsufficientIncentiveBalance();
95      }
96
97      balances[user] -= amount;
98      lockedPool += (amount + incentiveAmount);
99      incentivePool -= incentiveAmount;
100 }
```

Recommendation

Change to:

```
80  function lockTokensFrom(
81      address user,
82      uint256 amount,
83      uint256 incentiveAmount
84  ) public onlyBondingManager {
85      if (amount == 0 && incentiveAmount == 0) {
86          return;
```

```
87      }
88
89      if (balanceOf(user) < amount) {
90          revert InsufficientBondBalance();
91      }
92
93      if (incentivePool < incentiveAmount) {
94          revert InsufficientIncentiveBalance();
95      }
96
97      balances[user] -= amount;
98      lockedPool += (amount + incentiveAmount);
99      incentivePool -= incentiveAmount;
100 }
```

Status

✓ Fixed

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.