Report

v. 1.0

Customer

Limitless

Smart Contract Audit

Protocol V1

5th April 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog **6**

2 Introduction **7**

3 Project scope **8**

4 Methodology **9**

5 Our findings **10**

6 Major Issues **11**
CVF-1. INFO 11
CVF-2. INFO 11
CVF-3. INFO 11
CVF-4. INFO 12
CVF-5. INFO 12
CVF-6. INFO 12
CVF-7. INFO 13
CVF-8. INFO 13
CVF-9. INFO 14
CVF-10. INFO 14
CVF-11. INFO 15
CVF-12. INFO 15
CVF-13. INFO 15
CVF-14. INFO 16
CVF-15. INFO 16
CVF-16. INFO 16
CVF-17. INFO 17

7 Moderate Issues **18**
CVF-18. INFO 18
CVF-19. INFO 18
CVF-20. INFO 19
CVF-21. INFO 19
CVF-22. INFO 19
CVF-23. INFO 20
CVF-24. INFO 20
CVF-25. INFO 20
CVF-26. INFO 21
CVF-27. INFO 21
CVF-28. INFO 21
CVF-29. INFO 22
CVF-30. INFO 22
CVF-31. INFO 22

CVF-32. INFO 24

8 Recommendations **25**
CVF-33. INFO 25
CVF-34. INFO 25
CVF-35. INFO 26
CVF-36. INFO 26
CVF-37. INFO 26
CVF-38. INFO 27
CVF-39. INFO 27
CVF-40. INFO 27
CVF-41. INFO 28
CVF-42. INFO 28
CVF-43. INFO 28
CVF-44. INFO 28
CVF-45. INFO 29
CVF-46. INFO 29
CVF-47. INFO 29
CVF-48. INFO 30
CVF-49. INFO 30
CVF-50. INFO 30
CVF-51. INFO 31
CVF-52. INFO 31
CVF-53. INFO 32
CVF-54. INFO 32
CVF-55. INFO 32
CVF-56. INFO 33
CVF-57. INFO 33
CVF-58. INFO 33
CVF-59. INFO 34
CVF-60. INFO 34
CVF-61. INFO 34
CVF-62. INFO 35
CVF-63. INFO 35
CVF-64. INFO 35
CVF-65. INFO 36
CVF-66. INFO 36
CVF-67. INFO 36
CVF-68. INFO 37
CVF-69. INFO 38
CVF-70. INFO 38
CVF-71. INFO 39
CVF-72. INFO 40
CVF-73. INFO 40
CVF-74. INFO 41
CVF-75. INFO 41

CVF-76. INFO 41
CVF-77. INFO 42
CVF-78. INFO 42
CVF-79. INFO 42
CVF-80. INFO 42
CVF-81. INFO 43
CVF-82. INFO 43
CVF-83. INFO 43
CVF-84. INFO 44
CVF-85. INFO 44
CVF-86. INFO 44
CVF-87. INFO 45
CVF-88. INFO 45
CVF-89. INFO 45
CVF-90. INFO 46
CVF-91. INFO 46
CVF-92. INFO 46
CVF-93. INFO 47
CVF-94. INFO 47
CVF-95. INFO 47
CVF-96. INFO 48
CVF-97. INFO 48
CVF-98. INFO 48
CVF-99. INFO 48
CVF-100. INFO 49
CVF-101. INFO 49
CVF-102. INFO 49
CVF-103. INFO 50
CVF-104. INFO 50
CVF-105. INFO 50
CVF-106. INFO 50
CVF-107. INFO 51
CVF-108. INFO 51
CVF-109. INFO 51
CVF-110. INFO 51
CVF-111. INFO 52
CVF-112. INFO 52
CVF-113. INFO 52
CVF-114. INFO 53
CVF-115. INFO 53
CVF-116. INFO 54
CVF-117. INFO 54
CVF-118. INFO 54
CVF-119. INFO 55
CVF-120. INFO 55
CVF-121. INFO 56

CVF-122. INFO 56
CVF-123. INFO 56
CVF-124. INFO 56
CVF-125. INFO 57
CVF-126. INFO 57
CVF-127. INFO 57
CVF-128. INFO 58
CVF-129. INFO 58
CVF-130. INFO 58
CVF-131. INFO 59
CVF-132. INFO 59
CVF-133. INFO 59
CVF-134. INFO 59
CVF-135. INFO 60
CVF-136. INFO 60
CVF-137. INFO 60
CVF-138. INFO 61
CVF-139. INFO 61
CVF-140. INFO 62
CVF-141. INFO 62
CVF-142. INFO 63
CVF-143. INFO 63
CVF-144. INFO 64

1 Changelog

#	Date	Author	Description
0.1	11.04.24	A. Zveryanskaya	Initial Draft
0.2	11.04.24	A. Zveryanskaya	Minor revision
1.0	11.04.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

After fixing the indicated issues the smart contracts should be re-audited.

3 Project scope

We were asked to review:

- Original Code

Files:

```
/
    Executioner.sol          Facility.sol          MarginFacility.sol
    MarginFacilityHelper.sol PoolManager.sol       PremiumComputer.sol
    Util.sol
```

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 17 major, and a few less important issues.

Fixed 0 out of 17 issues

6 Major Issues

CVF-1 INFO

- **Category** Bad datatype
- **Source** Util.sol

Recommendation The return type should be a enum.

```
41 ) return 1;
```

```
46     ) return 2;
```

```
48 else return 3;
```

CVF-2 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description Checking this condition on every loop iteration is waste of gas.

Recommendation Consider checking once before the loop, and then implementing two separate loops, one for token0 and another for token1.

```
60 if (getToken0) {
```

CVF-3 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description Performing a linear search here is inefficient.

Recommendation Consider requiring both sets of loans to be sorted by tick to make merging much more efficient.

```
156 for(uint k; k< loans1.length; k++){
        if(loans2[j].tick== mergedLoans[k].tick){
```

CVF-4 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description Assigning the whole structure here is redundant, as only two fields are changed.

Recommendation Consider assigning only the changed fields.

```
159 mergedLoans[k] = LiquidityLoan(
160     mergedLoans[k].tick,
    mergedLoans[k].liquidity + loans2[j].liquidity,
    0,
    mergedLoans[k].feeGrowthInside0LastX128,
    mergedLoans[k].feeGrowthInside1LastX128,
    mergedLoans[k].lastGrowth
);
```

CVF-5 INFO

- **Category** Suboptimal
- **Source** Util.sol

Recommendation Allocating a new array here is redundant, as it is possible to trim an existing array using assembly.

```
184 LiquidityLoan[] memory returnedMergedLoans = new LiquidityLoan[](
    ↪ newLength);
```

CVF-6 INFO

- **Category** Suboptimal
- **Source** PremiumComputer.sol

Recommendation In case precision^2 fits into 256 bits, the first “mulDiv” call is an overkill and should be replaced with plain division.

```
127 multiplier = precision.mulDiv(precision, tickDiffInWad.mulDiv(scale,
    ↪  precision) );
```

CVF-7 INFO

- **Category** Flaw
- **Source** Facility.sol

Description The storage is updated after an external call.

Recommendation Consider doing external calls after all storage updates to avoid possible reentrancy attacks.

```
85 SafeTransferLib.safeTransferFrom(
```

```
88 PremiumDeposit[getPositionId( pool, msg.sender, payToken1)] +=
   ↪ depositAmount;
```

CVF-8 INFO

- **Category** Documentation
- **Source** PoolManager.sol

Description Reference to a particular constant value in a comment may become incorrect once the constant value would be changed.

Recommendation Consider referring to constants by names even in comments.

```
211 // at default, need to leave at least 100 units of liquidty
minLiquidities[hashedKey] = MINIMUM_LIQUIDITY;
```

CVF-9 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Description Transferring two different tokens in a single transaction is a bad practice, as in case one transfer fails, another one wouldn't be executed as well.

Recommendation Consider counting in the storage how many various tokens each user is eligible to withdraw, and allow users tow withdraw each token separately.

```
300 if (vars.totalLPFee0 > 0) TransferHelper.safeTransfer(key.token0,
    ↪ owner, vars.totalLPFee0);
    if (vars.totalLPFee1 > 0) TransferHelper.safeTransfer(key.token1,
    ↪ owner, vars.totalLPFee1);
```

```
377 if (vars.totalAmount0 > 0) TransferHelper.safeTransfer(key.token0,
    ↪ msg.sender, vars.totalAmount0);
    if (vars.totalAmount1 > 0) TransferHelper.safeTransfer(key.token1,
    ↪ msg.sender, vars.totalAmount1);
    if (vars.totalLPFee0 > 0) TransferHelper.safeTransfer(key.token0,
    ↪ owner, vars.totalLPFee0);
380 if (vars.totalLPFee1 > 0) TransferHelper.safeTransfer(key.token1,
    ↪ owner, vars.totalLPFee1);
```

```
507 if (totalLPFee0 > 0) TransferHelper.safeTransfer(key.token0, owner,
    ↪ totalLPFee0);
    if (totalLPFee1 > 0) TransferHelper.safeTransfer(key.token1, owner,
    ↪ totalLPFee1);
```

CVF-10 INFO

- **Category** Flaw
- **Source** MarginPositionHelper.sol

Description There is no check to ensure these arrays are of the same length. If "repayInfo" is longer than "borrowInfoCopy", then extra elements of "repayInfo" are silently ignored.

Recommendation Consider performing proper length check.

```
52 LiquidityLoan[] memory repayInfo,
   LiquidityLoan[] memory borrowInfoCopy
```

CVF-11 INFO

- **Category** Readability
- **Source** MarginPositionHelper.sol

Description Using both “int” and ”int256” in the same file makes code harder to read.

Recommendation Consider sticking to one notation.

```
188 if(amount0 < 0) amount0 -= int(marginInPosToken);
    else amount1 -= int(marginInPosToken);
```

CVF-12 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Description Transferring two different tokens in a single transaction is a bad practice, as in case one transfer fails, another one wouldn’t be executed as well.

Recommendation Consider counting in the storage how many various tokens each user is eligible to withdraw, and allow users tow withdraw each token separately.

```
220 SafeTransferLib.safeTransfer(ERC20(param.positionIsToken0 ? key.
       ↪ token1 : key.token0), poolManager, vars.feeToPool);
    SafeTransferLib.safeTransfer(ERC20(param.positionIsToken0 ? key.
       ↪ token1 : key.token0), feeReceiver,
         vars.fee-vars.feeToPool);
```

```
267 SafeTransferLib.safeTransfer(ERC20(param.positionIsToken0 ? key.
       ↪ token1 : key.token0), poolManager, vars.feeToPool);
    SafeTransferLib.safeTransfer(ERC20(param.positionIsToken0 ? key.
       ↪ token1 : key.token0), feeReceiver,
         vars.fee-vars.feeToPool);
```

CVF-13 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Recommendation This TODO should be resolved or removed.

```
379 //TODO test whether slippage makes this unfeasible?
```

CVF-14 INFO

- **Category** Readability
- **Source** MarginPositionHelper.sol

Description Using both “uint” and ”uint256” in the same file makes code harder to read.

Recommendation Consider sticking to one notation.

```
382 uint buyAmount = param.positionIsToken0? localVars.amount1Required:
       ↪ localVars.amount0Required;
    uint orderPrice = vars.reduceAmount.mulDiv(precision, vars.
       ↪ fillerOutput);
    uint sellAmount = buyAmount.mulDiv(orderPrice, precision);
```

CVF-15 INFO

- **Category** Suboptimal
- **Source** MarginFacility.sol

Description The whole “MarginPosition” structure is loaded into memory, while only one field is actually used.

Recommendation Consider loading only the actually used fields.

```
 79 MarginPosition memory pos = positions[getPositionId(pool, borrower,
       ↪ borrowedToken1)];
 80 return pos.base.borrowInfo;
```

CVF-16 INFO

- **Category** Suboptimal
- **Source** MarginFacility.sol

Description Pushing elements into an in-storage array one by one is inefficient.

Recommendation Consider preparing an in-memory array and then assigning it to the in-storage array at once.

```
121 pos.base.borrowInfo.push(borrowInfo[i]);
```

CVF-17 INFO

- **Category** Bad datatype
- **Source** MarginFacility.sol

Recommendation The valid values of “rangeCondition” should be named constants.

```
285 if (rangeCondition == 1) {
```

```
303 } else if (rangeCondition == 2) {
```

```
308     if(param.executionOption==1 || param.executionOption==2){
```

```
363 if (rangeCondition == 1) {
```

```
373 } else if (rangeCondition == 2) {
```

7 Moderate Issues

CVF-18 INFO

- **Category** Overflow/Underflow
- **Source** Util.sol

Description Overflow is possible during conversion to “uint128”.

Recommendation Consider using safe conversion.

```
87    uint128(FullMath.mulDiv(uint256(borrowInfo[i].liquidity),
          ↪ reducePercentage, precision));
```

```
100 uint128 liquidity = uint128(FullMath.mulDiv(uint256(borrowInfo[i].
      ↪ liquidity), percentageClosed, precision));
```

```
107             int128(liquidity)
```

```
123             int128(liquidity)
```

```
132             int128(liquidity)
```

CVF-19 INFO

- **Category** Documentation
- **Source** Util.sol

Description In case “tick” is a factor of “tickSpacing”, the returned value could differ from “tick” which seems incorrect.

Recommendation Consider either fixing or clearly documenting this behavior.

```
209 if (tick > 0) return down ? index * tickSpacing : ((index + 1) * (
      ↪ tickSpacing));
210 else return down ? index * tickSpacing - tickSpacing : (index * (
      ↪ tickSpacing));
```

CVF-20 INFO

- **Category** Procedural
- **Source** PremiumComputer.sol

Recommendation This TODO should be resolved.

```
41 // TODO maybe exploited
```

CVF-21 INFO

- **Category** Overflow/Underflow
- **Source** PremiumComputer.sol

Description Underflow is possible during type conversion.

Recommendation Consider using save conversion.

```
55 return int24( (tickCumulatives[1] - tickCumulatives[0])/int56(int(
   ↪ interval)));
```

CVF-22 INFO

- **Category** Overflow/Underflow
- **Source** PremiumComputer.sol

Description Overflow is possible during type conversion.

Recommendation Consider using safe conversion.

```
277 vars.twat = getTWAT( pool,  block.timestamp-uint(
    ↪ lastPremiumPaymentTime));
```

CVF-23 INFO

- **Category** Overflow/Underflow
- **Source** PoolManager.sol

Description Overflow is possible when converting types.

Recommendation Consider using save conversions or changing the types of the arguments to make conversion unnecessary.

```
94 feeParams[hashedKey].openFee = uint128(openFee);
   feeParams[hashedKey].feeToPool = uint128(feeToPool);
   feeParams[hashedKey].profitFee = uint128(profitFee);
   feeParams[hashedKey].profitFeeToPool = uint128(profitFeeToPool);
```

CVF-24 INFO

- **Category** Unclear behavior
- **Source** PoolManager.sol

Description This should be done only when “liquidity” is greater than zero.

```
291 binBitmaps[vars.hashedKey].flipTick(tickLower_, vars.
      ↪ tickDiscretization);
```

CVF-25 INFO

- **Category** Overflow/Underflow
- **Source** PoolManager.sol

Description Overflow is possible during type conversion.

Recommendation Consider using safe conversion.

```
533 vars.totalFees0 = uint128(
```

```
538 vars.totalFees1 = uint128(
```

CVF-26 INFO

- **Category** Overflow/Underflow
- **Source** PoolManager.sol

Description Overflow is possible during type conversion.

Recommendation Consider using safe conversion.

```
570 balance = uint128(ERC20(key.token0).balanceOf(address(this)));
```

```
576 balance = uint128(ERC20(key.token1).balanceOf(address(this)));
```

CVF-27 INFO

- **Category** Overflow/Underflow
- **Source** PoolManager.sol

Description Overflow is possible during type conversions.

Recommendation Consider using safe conversion.

```
636 uint128(amount0),
    uint128(amount1)
```

CVF-28 INFO

- **Category** Overflow/Underflow
- **Source** Executioner.sol

Description Overflow is possible during type conversions.

Recommendation Consider using safe conversions.

```
37 outIsToken0 ? (-int256(outputAmount), int256(inputAmount)) : (int256
   ↪ (inputAmount), -int256(outputAmount));
```

CVF-29 INFO

- **Category** Overflow/Underflow
- **Source** MarginPositionHelper.sol

Description Overflow is possible when converting types.

Recommendation Consider using safe conversion.

```
33      pos.base.openTime = uint32(block.timestamp);
```

```
35      pos.base.lastPremiumPaymentTime = uint32(block.timestamp);
```

```
79  order.auctionDeadline = uint32(deadline);
80  order.auctionStartTime = uint32(block.timestamp);
```

CVF-30 INFO

- **Category** Overflow/Underflow
- **Source** MarginPositionHelper.sol

Description Overflow is possible during type conversions.

Recommendation Consider using safe conversion.

```
173         marginInPosToken == 0? int256(param.margin + param.
               ↪ borrowAmount) : int256(param.borrowAmount) ,
```

```
188 if(amount0 < 0) amount0 -= int(marginInPosToken);
    else amount1 -= int(marginInPosToken);
```

```
192 require((param.positionIsToken0 ? uint256(-amount0) : uint256(-
       ↪ amount1)) >= filledAmount, "!avgPrice");
```

CVF-31 INFO

- **Category** Overflow/Underflow
- **Source** MarginPositionHelper.sol

Description Over-/underflow is possible when converting types.

Recommendation Consider using safe conversion.

```
376         addresses.marginInPosToken ? param.positionIsToken0 ? -
               ↪ int256(localVars.amount1Required) : -int256(localVars.
```

```
            ↪ amount0Required)
             : int256(vars.reduceAmount), key.token0, key.token1,
                ↪ param.minOutput
```

```
412     if (uint256(-returnVars.amount1) < returnVars.repaidDebt1)
           ↪ revert Errors.wrongAmountRepay();
```

```
416         : -returnVars.amount1 - int256(returnVars.repaidDebt1);
```

```
419     if (uint256(-returnVars.amount0) < returnVars.repaidDebt0)
           ↪ revert Errors.wrongAmountRepay();
```

```
421         ? int(vars.reduceAmount) - returnVars.amount1
            : -returnVars.amount0 - int256(returnVars.repaidDebt0);
```

```
427 localVars.marginReduced = int256(vars.margin.mulDiv(param.
       ↪ reducePercentage, precision));
```

```
431     returnVars.profitFee = uint256(returnVars.PnL -localVars.
           ↪ marginReduced).mulDiv(localVars.profitFeePercentage,
           ↪ precision);
```

```
438 SafeTransferLib.safeTransfer(ERC20(localVars.token), param.trader,
       ↪ uint256(returnVars.PnL) - returnVars.profitFee);
```

```
441 returnVars.PnL -= (localVars.marginReduced + int256(returnVars.
       ↪ profitFee));
```

```
462         vars.pool, param.positionIsToken0, -int256(requiredOutput),
               ↪ key.token0, key.token1, 0
```

```
482     require(uint256(-returnVars.amount1) >= returnVars.repaidDebt1,
           ↪ "IRwrongAmountRepay");
        require(vars.reduceAmount >= uint(returnVars.amount0), "
           ↪ IRWrongSwap");
        require(returnVars.repaidDebt0< vars.reduceAmount - uint(
           ↪ returnVars.amount0), "IRinsolvent");
```

```
486     require(uint256(-returnVars.amount0) >= returnVars.repaidDebt0,
           ↪ "IRwrongAmountRepay");
        require(vars.reduceAmount >= uint(returnVars.amount1), "
           ↪ IRWrongSwap");
        require(returnVars.repaidDebt1< vars.reduceAmount - uint(
           ↪ returnVars.amount1), "IRinsolvent");
```

```
493         ? uint(-returnVars.amount1) - returnVars.repaidDebt1
            : uint(-returnVars.amount0) - returnVars.repaidDebt0;
```

```
505         ? int(vars.reduceAmount) - returnVars.amount0 - int(
               ↪ returnVars.repaidDebt0) - localVars.amount0
            : int(vars.reduceAmount) - returnVars.amount1 - int(
               ↪ returnVars.repaidDebt1) - localVars.amount1;
```

```
508     SafeTransferLib.safeTransfer(ERC20(param.positionIsToken0 ? key.
           ↪ token0 : key.token1), param.trader, uint256(returnVars.PnL
           ↪ ) );
```

```
511         ? vars.reduceAmount - uint(returnVars.amount0)- returnVars.
               ↪ repaidDebt0
            : vars.reduceAmount - uint(returnVars.amount1)- returnVars.
               ↪ repaidDebt1;
```

```
607 returnVars.PnL = -int256(margin.mulDiv(param.reducePercentage,
       ↪ precision));
```

```
707         ? vars.amount1Required > uint256(-vars.amount1)
            : vars.amount0Required > uint256(-vars.amount0)
```

```
723         < (positionIsToken0 ? uint256(vars.amount0) + vars.
               ↪ repaidDebt0 : uint256(vars.amount1) + vars.repaidDebt1
               ↪ )
```

```
729         - (positionIsToken0 ? uint256(vars.amount0) + vars.
               ↪ repaidDebt0 : uint256(vars.amount1) + vars.repaidDebt1
               ↪ );
```

CVF-32 INFO

- **Category** Procedural
- **Source** MarginFacility.sol

Recommendation This comment looks like a TODO. The mentioned attacks should be addressed.

```
450 * possible attacks
*   - filler manipulate slipped ticks when reducing
*   - filler submits very expensive borrowinfo when adding position
```

8 Recommendations

CVF-33 INFO

- **Category** Procedural
- **Source** Util.sol

Description Specifying compiler version range without upper bound is discouraged, as it is not possible to guarantee compatibility with future major versions.

Recommendation Consider specifying as "^0.8.0" or "^0.8.8" if there is something special regarding this particular version. Also relevant for: PremiumComputer.sol, Facility.sol, PoolManager.sol, Executioner.sol, MarginPositionHelper.sol, MarginFacility.sol.

```
1 pragma solidity >=0.8.8;
```

CVF-34 INFO

- **Category** Procedural
- **Source** Util.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
3 import {IUniswapV3Pool} from "./uniswap/interfaces/IUniswapV3Pool.
    ↪ sol";
  import {SqrtPriceMath} from "./uniswap/SqrtPriceMath.sol";
  import {TickMath} from "./uniswap/TickMath.sol";

7 import {FullMath} from "./uniswap/FullMath.sol";
  import {LiquidityAmounts} from "./uniswap/LiquidityAmounts.sol";

10 import {FixedPoint96} from "./uniswap/FixedPoint96.sol";
   import {SafeCast} from "./uniswap/SafeCast.sol";
```

CVF-35 INFO

- **Category** Suboptimal
- **Source** Util.sol

Recommendation These two imports should be merged.

```
6 import {LiquidityLoan} from "./types.sol";
```

```
9 import {precision} from "./types.sol";
```

CVF-36 INFO

- **Category** Procedural
- **Source** Util.sol

Description We didn't review this file.

```
6 import {LiquidityLoan} from "./types.sol";
```

CVF-37 INFO

- **Category** Documentation
- **Source** Util.sol

Description The semantics and format of the returned value is unclear.

Recommendation Consider documenting.

```
24 ) public pure returns(uint){
```

```
37 ) public pure returns(uint){
```

CVF-38 INFO

- **Category** Unclear behavior
- **Source** Util.sol

Description This description of returned values is unclear and doesn't match with the code. According to the code, 1 means above range, 2 means below range, and 3 means inside range.

```
29 /// @notice for given curTick, checks if the borrowed ticks are out
       ↪ of range(1),
30 /// in range(2), or totally crossed range(3)
```

CVF-39 INFO

- **Category** Bad datatype
- **Source** Util.sol

Recommendation These values should be named constants.

```
41 ) return 1;
```

```
46     ) return 2;
```

```
48 else return 3;
```

CVF-40 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description The expression "borrowInfo[i].liquidity" is calculated several times.

Recommendation Consider calculating once and reusing.

```
59 if (borrowInfo[i].liquidity == 0) continue;
```

```
65             -int256(uint256(borrowInfo[i].liquidity)).toInt128()
```

```
73             -int256(uint256(borrowInfo[i].liquidity)).toInt128()
```

CVF-41 INFO

- **Category** Procedural
- **Source** Util.sol

Description This comment is confusing.

Recommendation Consider removing it.

```
208 int24 index = tick / tickSpacing; //1820/100 = 30
```

CVF-42 INFO

- **Category** Procedural
- **Source** Util.sol

Recommendation Brackets around “tickSpacing” and around multiplication are redundant.

```
209 if (tick > 0) return down ? index * tickSpacing : ((index + 1) * (
    ↪ tickSpacing));
```

CVF-43 INFO

- **Category** Procedural
- **Source** Util.sol

Recommendation Brackets are redundant.

```
210 else return down ? index * tickSpacing - tickSpacing : (index * (
    ↪ tickSpacing));
```

CVF-44 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description In case precision^2 fit into 256 bits, using “mulDiv” here is an overkill.

Recommendation Consider using simple division.

```
215 maxSlippage = down ? FullMath.mulDiv(precision, precision,
    ↪ maxSlippage) : maxSlippage;
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description It would be more efficient to replace multiplication with a shift for this calculation.

Recommendation Consider implementing “shlDiv” function.

```
216 uint256 slippage = FullMath.mulDiv(maxSlippage, FixedPoint96.Q96,
    ↪ precision);
```

CVF-46 INFO

- **Category** Suboptimal
- **Source** Util.sol

Recommendation Type conversions are redundant.

```
217 return toUint160(FullMath.mulDiv(uint256(price), slippage, uint256(
    ↪ FixedPoint96.Q96)));
```

CVF-47 INFO

- **Category** Suboptimal
- **Source** Util.sol

Description It would be more efficient to replace division by a shift for this calculation.

Recommendation Consider implementing “mulShr” function.

```
217 return toUint160(FullMath.mulDiv(uint256(price), slippage, uint256(
    ↪ FixedPoint96.Q96)));
```

CVF-48 INFO

- **Category** Suboptimal
- **Source** PremiumComputer.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
5  import {IUniswapV3Pool} from "./uniswap/interfaces/IUniswapV3Pool.
       ↪ sol";
   import {TickMath} from "./uniswap/TickMath.sol";
   import {LiquidityAmounts} from "./uniswap/LiquidityAmounts.sol";
   import {FullMath} from "./uniswap/FullMath.sol";

10 import {TransferHelper} from "./uniswap/TransferHelper.sol";
   import {FixedPoint128} from "./uniswap/FixedPoint128.sol";
```

CVF-49 INFO

- **Category** Procedural
- **Source** PremiumComputer.sol

Description We didn't review these files.

```
9  import {ERC20} from "./ERC20.sol";
```

```
13 import "./types.sol";
```

```
16 import {SafeTransferLib} from "./utils/SafeTransferLib.sol";
```

CVF-50 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The argument type should be "IUniswapV3Pool".

```
24 function getOldestObservationSecondsAgo(address pool) internal view
       ↪ returns (uint32 secondsAgo) {
```

CVF-51 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The type for the “pool” argument should be “IUniswapV3Pool”.

```
40 function getTWAT(address pool, uint256 interval) public view returns
   ↪ (int24){
```

```
92     address pool,
```

```
141    address pool,
```

```
267    address pool,
```

CVF-52 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The value “1e18” should be a named constant.

```
60  require(uRate <= 1e18, 'MAX_UTILIZED');
```

```
125 uint tickDiffInWad =  uint(int(tickDiff)) * 1e18;
```

CVF-53 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The type for this argument should be “PoolManager”.

```
93  address poolManager,
```

```
142 address poolManager,
```

```
190 address poolManager,
```

```
214 address poolManager,
```

```
237 address poolManager,
```

```
268 address poolManager,
```

```
336 address poolManager,
```

CVF-54 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The value “10e18” should be a named constant.

```
123 if(tickDiff==0) return 10e18;
```

CVF-55 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The value “31536000” should be a named constant.

```
168     uint interest = (precision+getInterestRate(param, urate)).rpow
          ↪ (31536000, precision ) - precision;
```

```
183 uint interest = (precision+getInterestRate(param, urate)).rpow
      ↪ (31536000, precision ) - precision;
```

CVF-56 INFO

- **Category** Bad datatype
- **Source** PremiumComputer.sol

Recommendation The type for this argument should be “Executioner”.

191
```
address executioner,
```

337
```
address executioner
```

CVF-57 INFO

- **Category** Unclear behavior
- **Source** PremiumComputer.sol

Description In case “isClose” is true, the “amount” value is ignored, which seems odd.

Recommendation Consider removing the “isClose” argument and just using a special “amount” value to indicate and intent to close the account.

240
```
uint256 amount,
bool isClose
```

CVF-58 INFO

- **Category** Suboptimal
- **Source** PremiumComputer.sol

Description The actual withdrawn amount could differ from this value in case “isClose” is true.

Recommendation Consider returning the actual amount withdrawn.

240
```
uint256 amount,
```

CVF-59 INFO

- **Category** Procedural
- **Source** PremiumComputer.sol

Recommendation This should be placed once after the conditional operator.

```
311 // total premium is swap fee + interest
    vars.totalFee += vars.interest;
```

```
320 vars.totalFee += vars.interest;
```

CVF-60 INFO

- **Category** Bad naming
- **Source** PremiumComputer.sol

Recommendation Events are usually named via nouns, such as “PremiumDeposit” or “PremiumPayment”.

```
443 event PremiumDeposited(
```

```
450 event PremiumPaid(
```

CVF-61 INFO

- **Category** Procedural
- **Source** Facility.sol

Description We didn’t review these files.

```
5 import "./types.sol";
```

```
7 import {ERC20} from "./ERC20.sol";
  import {SafeTransferLib} from "./utils/SafeTransferLib.sol";
```

```
12 import {IFacility} from "./interfaces/IFacility.sol";
```

CVF-62 INFO

- **Category** Procedural
- **Source** Facility.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
6 import {FullMath} from "./uniswap/FullMath.sol";
```

CVF-63 INFO

- **Category** Unclear behavior
- **Source** Facility.sol

Description These functions should emit some events.

```
44 function setOwner(address newOwner) external onlyOwner {
```

```
48 function setProtocolContracts(address poolManager_, address
   ↪ executioner_) external onlyOwner {
```

CVF-64 INFO

- **Category** Bad datatype
- **Source** Facility.sol

Recommendation The argument types should be “PoolManager” and “Executioner” respectively.

```
48 function setProtocolContracts(address poolManager_, address
   ↪ executioner_) external onlyOwner {
```

CVF-65 INFO

- **Category** Bad datatype
- **Source** Facility.sol

Recommendation The argument types should be “PoolManager” and “Executioner” respectively.

```
58 function _initialize(address pm, address ex) public onlyInitializing
   ↪  {
```

CVF-66 INFO

- **Category** Documentation
- **Source** Facility.sol

Description The semantics of the returned values is unclear.

Recommendation Consider documenting.

```
96 function _payPremium(PayParams memory param) internal returns (
   ↪ uint256, LiquidityLoan[] memory) {
```

CVF-67 INFO

- **Category** Bad datatype
- **Source** Facility.sol

Recommendation The type for the arguments should be “IERC20”.

```
140 function approveTokens(address token0, address token1) external {
```

CVF-68 INFO

- **Category** Bad datatype
- **Source** Facility.sol

Recommendation The type for the “pool” argument should be more specific.

```
149 function getPositionId(address pool, address trader, bool
    ↪ positionIsToken0) public pure returns (PositionId) {

152 function toId(address pool, address trader, bool positionIsToken0)
    ↪ internal pure returns (PositionId) {

156 function getOrderId(address pool, address trader, bool
    ↪ positionIsToken0, bool isAdd)

164 event PremiumWithdrawn(address indexed withdrawer, address indexed
    ↪ pool, bool isToken1, uint256 amount);

166 event PremiumDeposited(address indexed payer, address indexed pool,
    ↪ bool isToken1, uint256 amount);

168 function _updateBorrowInfo(address pool, address borrower, bool
    ↪ borrowedToken1, LiquidityLoan[] memory)

171 function _updateLastPremiumPaymentTime(address pool, address
    ↪ borrower, bool borrowedToken1) internal virtual;

173 function checkPositionExists(address pool, address borrower, bool
    ↪ borrowedToken1)

179 function getBorrowInfo(address pool, address borrower, bool
    ↪ borrowedToken1)

184 function getLastRepayTime(address pool, address borrower, bool
    ↪ borrowedToken1)
```

CVF-69 INFO

- **Category** Procedural
- **Source** PoolManager.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
4  import {TickBitmap} from "./uniswap/TickBitmap.sol";
   import {IUniswapV3Pool} from "./uniswap/interfaces/IUniswapV3Pool.
      ↪ sol";
   import {TickMath} from "./uniswap/TickMath.sol";
   import {LiquidityAmounts} from "./uniswap/LiquidityAmounts.sol";
   import {FullMath} from "./uniswap/FullMath.sol";

10 import {TransferHelper} from "./uniswap/TransferHelper.sol";
   import {FixedPoint128} from "./uniswap/FixedPoint128.sol";

21 import {CallbackValidation} from "./uniswap/CallbackValidation.sol";
```

CVF-70 INFO

- **Category** Procedural
- **Source** PoolManager.sol

Description We didn't review these files.

```
9  import {ERC20} from "./ERC20.sol";

13 import "./types.sol";
   import {MatchingEngine} from "./MatchingEngine.sol";

17 import {IPoolManager} from "./interfaces/IPoolManager.sol";

19 import {SafeTransferLib} from "./utils/SafeTransferLib.sol";
```

CVF-71 INFO

- **Category** Documentation
- **Source** PoolManager.sol

Description The semantics of keys and values in these mappings is unclear.

Recommendation Consider documenting.

```
27 mapping(bytes32 => mapping(int24 => uint128)) public LiquidityBin;
   mapping(bytes32 => mapping(int24 => uint128)) public
      ↪ borrowedLiquidities;
   mapping(bytes32 => mapping(int24 => Interest)) public Interests;
30 mapping(bytes32 => mapping(bytes32 => UniswapPosition)) public
      ↪ LiquidityPositions;
   mapping(address => mapping(int24 => UtilizationGrowth)) public
      ↪ UtilizationGrowths;
```

```
34 mapping(address => PoolParam) public PoolParams;
   mapping(bytes32 => uint256) public minLiquidities;
   mapping(bytes32 => FeeParams) public feeParams;
   mapping(bytes32 => int24) public tickDiscretizations;
```

```
39 mapping(address => mapping(address => mapping(uint24 => address)))
      ↪ public getPool;
40 mapping(bytes32 => mapping(int16 => uint256)) public binBitmaps;
```

```
44 mapping(uint256 => address) internal _poolList;
```

```
54 mapping(address=> bool) public poolTrusted;
```

CVF-72 INFO

- **Category** Bad naming
- **Source** PoolManager.sol

Recommendation Storage variable names usually start with lower case letters.

```
27 mapping(bytes32 => mapping(int24 => uint128)) public LiquidityBin;
```

```
29 mapping(bytes32 => mapping(int24 => Interest)) public Interests;
30 mapping(bytes32 => mapping(bytes32 => UniswapPosition)) public
      ↪ LiquidityPositions;
   mapping(address => mapping(int24 => UtilizationGrowth)) public
      ↪ UtilizationGrowths;
```

```
34 mapping(address => PoolParam) public PoolParams;
```

CVF-73 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are hashed keys and values are structs encapsulating the values of the original mappings.

```
27 mapping(bytes32 => mapping(int24 => uint128)) public LiquidityBin;
   mapping(bytes32 => mapping(int24 => uint128)) public
      ↪ borrowedLiquidities;
   mapping(bytes32 => mapping(int24 => Interest)) public Interests;
30 mapping(bytes32 => mapping(bytes32 => UniswapPosition)) public
      ↪ LiquidityPositions;
```

```
35 mapping(bytes32 => uint256) public minLiquidities;
   mapping(bytes32 => FeeParams) public feeParams;
   mapping(bytes32 => int24) public tickDiscretizations;
```

```
40 mapping(bytes32 => mapping(int16 => uint256)) public binBitmaps;
```

CVF-74 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are hashed keys and lower ticks, and values are structs encapsulating the values of the original mappings.

```
27 mapping(bytes32 => mapping(int24 => uint128)) public LiquidityBin;
   mapping(bytes32 => mapping(int24 => uint128)) public
      ↪ borrowedLiquidities;
   mapping(bytes32 => mapping(int24 => Interest)) public Interests;
```

CVF-75 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are pools and values are structs encapsulating the values of the original mappings.

```
31 mapping(address => mapping(int24 => UtilizationGrowth)) public
      ↪ UtilizationGrowths;
```

```
34 mapping(address => PoolParam) public PoolParams;
```

```
54 mapping(address=> bool) public poolTrusted;
```

CVF-76 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for the first key should be more specific.

```
31 mapping(address => mapping(int24 => UtilizationGrowth)) public
      ↪ UtilizationGrowths;
```

```
34 mapping(address => PoolParam) public PoolParams;
```

```
54 mapping(address=> bool) public poolTrusted;
```

CVF-77 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The value type should be more specific.

```
39 mapping(address => mapping(address => mapping(uint24 => address)))
   ↪ public getPool;
```

```
44 mapping(uint256 => address) internal _poolList;
```

CVF-78 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for the first two keys should be “IERC20”.

```
39 mapping(address => mapping(address => mapping(uint24 => address)))
   ↪ public getPool;
```

CVF-79 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for this variable should be “Facility”.

```
48 address public marginFacility;
```

CVF-80 INFO

- **Category** Procedural
- **Source** PoolManager.sol

Description Similar modifier is declared in other contracts.

Recommendation Consider extracting into a library or into a common base contract.

```
56 modifier nonReentrant() virtual {
```

CVF-81 INFO

- **Category** Unclear behavior
- **Source** PoolManager.sol

Description These functions should emit some events.

```
77 function setOwner(address newOwner) external onlyOwner {
```

```
81 function setFacilities(address mf) external onlyOwner {
```

```
85 function updateFeeParams(
```

```
101 function updatePoolParams(address pool, PoolParam memory param)
      ↪ external onlyOwner {
```

CVF-82 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The argument type should be “Facility”.

```
81 function setFacilities(address mf) external onlyOwner {
```

CVF-83 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Description The expression “feeParams[hashedKey]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
94 feeParams[hashedKey].openFee = uint128(openFee);
   feeParams[hashedKey].feeToPool = uint128(feeToPool);
   feeParams[hashedKey].profitFee = uint128(profitFee);
   feeParams[hashedKey].profitFeeToPool = uint128(profitFeeToPool);
   feeParams[hashedKey].LPFee = LPFee;
```

CVF-84 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for the “pool” arguments should be more specific.

```
101 function updatePoolParams(address pool, PoolParam memory param)
    ↪ external onlyOwner {
```

```
123 function getGrowth(address pool, int24 tick, URateParam memory param
    ↪ ) public view returns (uint256) {
```

```
188     address pool,
```

```
411 function _updateUtilizationGrowth(int24 tick, bytes32 hashedKey,
    ↪ address pool, URateParam memory param) internal {
```

CVF-85 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for the “pool” returned value should be more specific.

```
116 returns (address pool, int24 tickDiscretization, PoolParam memory
    ↪ param)
```

CVF-86 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The return type should be more specific.

```
155 function getPoolList() external view returns (address[] memory) {
```

CVF-87 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for the “pool” parameters should be more specific.

```
165 address indexed pool, address indexed recipient, uint128 liquidity,
    ↪ int24 tickLower, int24 tickUpper
```

```
169 address indexed pool, address indexed recipient, uint128 liquidity,
    ↪ int24 tickLower, int24 tickUpper
```

```
173 address indexed pool,
```

```
182 address indexed pool, address indexed token0, address indexed token1
    ↪ , uint24 fee, int24 tickDiscretization
```

CVF-88 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for the token parameters should be “IERC20”.

```
182 address indexed pool, address indexed token0, address indexed token1
    ↪ , uint24 fee, int24 tickDiscretization
```

CVF-89 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The type for these arguments should be “IERC20”.

```
189 address tokenA,
190 address tokenB,
```

CVF-90 INFO

- **Category** Procedural
- **Source** PoolManager.sol

Recommendation These checks should be done earlier.

```
197 require(tokenA != tokenB, "same token");
    require(token0 != address(0) && token1 != address(0), "zero address"
       ↪ );
    require(getPool[token0][token1][fee] == address(0), "pool exists");
200 require(getPool[token1][token0][fee] == address(0), "pool exists");
```

CVF-91 INFO

- **Category** Procedural
- **Source** PoolManager.sol

Recommendation This TODO should be resolved or removed.

```
204 // TODO callbackvalidations for permissionless listing
```

CVF-92 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Description The expression “feeParams[hashedKey]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
222 feeParams[hashedKey].openFee = uint128(1e15); //0.1%
    feeParams[hashedKey].feeToPool = uint128(5e17); // 50% of open fee
       ↪ to pool
    feeParams[hashedKey].profitFee = uint128(5e16); // 5% of profit as
       ↪ fee
    feeParams[hashedKey].profitFeeToPool = uint128(5e17); // 50% of
       ↪ profit fee to pool
    feeParams[hashedKey].LPFee = 1e17;
```

CVF-93 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation Type conversions are redundant here.

```
222 feeParams[hashedKey].openFee = uint128(1e15); //0.1%
    feeParams[hashedKey].feeToPool = uint128(5e17); // 50% of open fee
       ↪ to pool
    feeParams[hashedKey].profitFee = uint128(5e16); // 5% of profit as
       ↪ fee
    feeParams[hashedKey].profitFeeToPool = uint128(5e17); // 50% of
       ↪ profit fee to pool
```

CVF-94 INFO

- **Category** Readability
- **Source** PoolManager.sol

Recommendation This value could be rendered as “0.001e18”.

```
222 feeParams[hashedKey].openFee = uint128(1e15); //0.1%
```

CVF-95 INFO

- **Category** Readability
- **Source** PoolManager.sol

Recommendation This value could be rendered as “0.5e18”.

```
223 feeParams[hashedKey].feeToPool = uint128(5e17); // 50% of open fee
       ↪ to pool
```

```
225 feeParams[hashedKey].profitFeeToPool = uint128(5e17); // 50% of
       ↪ profit fee to pool
```

CVF-96 INFO

- **Category** Readability
- **Source** PoolManager.sol

Recommendation This value could be rendered as “0.05e18”.

```
224 feeParams[hashedKey].profitFee = uint128(5e16); // 5% of profit as
      ↪ fee
```

CVF-97 INFO

- **Category** Readability
- **Source** PoolManager.sol

Recommendation This value could be rendered as “0.1e18”.

```
226 feeParams[hashedKey].LPFee = 1e17;
```

CVF-98 INFO

- **Category** Bad datatype
- **Source** PoolManager.sol

Recommendation The value “100” should be a named constant.

```
229 IUniswapV3Pool(pool).increaseObservationCardinalityNext(100);
```

CVF-99 INFO

- **Category** Bad naming
- **Source** PoolManager.sol

Recommendation Variable names usually start with lower case letter.

```
247 IUniswapV3Pool UniPool = IUniswapV3Pool(getPool[key.token0][key.
      ↪ token1][key.fee]);
```

CVF-100 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation Type conversions are redundant.

```
404 interest.interest0 += FullMath.mulDiv(uint256(interestTokens0),
       ↪ FixedPoint128.Q128, uint256(liquidity));
    interest.interest1 += FullMath.mulDiv(uint256(interestTokens1),
       ↪ FixedPoint128.Q128, uint256(liquidity));
```

CVF-101 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation Type conversions are redundant.

```
449 vars.feePerTick = feePortion.mulDiv(uint256(borrowInfo[i].liquidity)
       ↪ , uint256(vars.totalLiquidity));
```

```
455     borrowInfo[i].premium + vars.feePerTick - vars.LPFee,
           ↪ FixedPoint128.Q128, uint256(vars.liquidity)
```

CVF-102 INFO

- **Category** Suboptimal
- **Source** PoolManager.sol

Recommendation Type conversions for “liquidity” are redundant.

```
502 interest.interest0 += FullMath.mulDiv(uint256(amount0) - fee0,
       ↪ FixedPoint128.Q128, uint256(liquidity));
    interest.interest1 += FullMath.mulDiv(uint256(amount1) - fee1,
       ↪ FixedPoint128.Q128, uint256(liquidity));
```

CVF-103 INFO

- **Category** Procedural
- **Source** Executioner.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
3 import {IUniswapV3Pool} from "./uniswap/interfaces/IUniswapV3Pool.
    ↪ sol";

5 import {TransferHelper} from "./uniswap/TransferHelper.sol";
  import {TickMath} from "./uniswap/TickMath.sol" ;
```

CVF-104 INFO

- **Category** Procedural
- **Source** Executioner.sol

Description We didn't review this file.

```
4 import {ERC20} from "./ERC20.sol";
```

CVF-105 INFO

- **Category** Bad datatype
- **Source** Executioner.sol

Recommendation The type for this variable should be "Facility".

```
12 address public marginFacility;
```

CVF-106 INFO

- **Category** Bad datatype
- **Source** Executioner.sol

Recommendation The type for this variable should be "PoolManager".

```
13 address public poolManager;
```

CVF-107 INFO

- **Category** Bad datatype
- **Source** Executioner.sol

Recommendation The argument types should be “Facility” and “PoolManager” respectively.

```
19 function setContracts(address _marginFacility, address _poolManager)
   ↪  external {
```

CVF-108 INFO

- **Category** Unclear behavior
- **Source** Executioner.sol

Description This function should emit some event.

```
19 function setContracts(address _marginFacility, address _poolManager)
   ↪  external {
```

CVF-109 INFO

- **Category** Bad datatype
- **Source** Executioner.sol

Recommendation The type for these arguments should be “IERC20”.

```
30 address token0,
   address token1
```

CVF-110 INFO

- **Category** Bad datatype
- **Source** Executioner.sol

Recommendation The type for this argument should be “IUniswapV3Pool”.

```
41 address pool,
```

CVF-111 INFO

- **Category** Bad datatype
- **Source** Executioner.sol

Recommendation The type for these arguments should be “IERC20”.

```
44 address token0,
   address token1,
```

CVF-112 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
4  import {IUniswapV3Pool} from "./uniswap/interfaces/IUniswapV3Pool.
       ↪ sol";
   import {TickMath} from "./uniswap/TickMath.sol";
   import {FullMath} from "./uniswap/FullMath.sol";
```

CVF-113 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Description We didn’t review this file.

```
7  import {ERC20} from "./ERC20.sol";
```

```
9  import "./types.sol";
```

```
14 import {SafeTransferLib} from "./utils/SafeTransferLib.sol";
```

CVF-114 INFO

- **Category** Suboptimal
- **Source** MarginPositionHelper.sol

Description The expression “pos.base” is calculated several times.

Recommendation Consider calculating once and reusing.

```
30  pos.base.totalDebtInput += param.borrowAmount;
    pos.base.totalDebtOutput += filledAmount;
    if (pos.base.openTime == 0) {
        pos.base.openTime = uint32(block.timestamp);
35      pos.base.lastPremiumPaymentTime = uint32(block.timestamp);
        pos.base.isToken0 = param.positionIsToken0;
40  if (pos.base.borrowInfo.length > 0) delete pos.base.borrowInfo;
43          pos.base.borrowInfo.push(borrowInfo[i]);
```

CVF-115 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Recommendation This commented line should be uncommented or removed.

```
34  // pos.base.repayTime = uint32(block.timestamp);
```

CVF-116 INFO

- **Category** Suboptimal
- **Source** MarginPositionHelper.sol

Description Pushing several elements into an in-storage array is inefficient, as the array length is updated multiple times.

Recommendation Consider preparing the array in memory and then assigning to the in-storage array an once.

```
40 if (pos.base.borrowInfo.length > 0) delete pos.base.borrowInfo;
   for (uint256 i; i < borrowInfo.length; i++) {
       if (borrowInfo[i].liquidity > 0) {
           pos.base.borrowInfo.push(borrowInfo[i]);
```

CVF-117 INFO

- **Category** Suboptimal
- **Source** MarginPositionHelper.sol

Recommendation It would be more efficient to pass a single array of structs with two elements, rather than two parallel arrays. This would also make the length check unnecessary.

```
52 LiquidityLoan[] memory repayInfo,
   LiquidityLoan[] memory borrowInfoCopy
```

CVF-118 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Description The expression “pos.base” is calculated several times.

Recommendation Consider calculating once and reusing.

```
56 pos.base.totalDebtInput = pos.base.totalDebtInput.mulDiv(precision -
      ↪  reducePercentage, precision);
   pos.base.totalDebtOutput = pos.base.totalDebtOutput.mulDiv(precision
      ↪  - reducePercentage, precision);
   pos.margin = pos.margin.mulDiv(precision - reducePercentage,
      ↪ precision);
```

```
63     pos.base.borrowInfo[i] = borrowInfoCopy[i];
```

CVF-119 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
92 address poolManager,
```

CVF-120 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The values of “executeOption” should be named constants.

```
136     require(param.executionOption==3, "!exec");
```

```
170 if (param.executionOption == 1) {
```

```
177 } else if (param.executionOption == 2) {} else if (param.
    ↪ executionOption == 3) {
```

```
331     require(param.executionOption == 3, "wrongExecution");
```

```
372 if (param.executionOption == 1) {
```

```
380 } else if (param.executionOption == 2) {} else if (param.
    ↪ executionOption == 3) {
```

```
396 if (param.executionOption == 1 || param.executionOption == 2) {
```

```
458 if (param.executionOption == 1) {
```

```
465 } else if (param.executionOption == 2) {}
```

```
468 if (param.executionOption == 1 || param.executionOption == 2) {
```

CVF-121 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “Executioner”.

```
165 address executioner,
```

CVF-122 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be more specific.

```
166 address pool,
```

CVF-123 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
201 address poolManager,
```

CVF-124 INFO

- **Category** Documentation
- **Source** MarginPositionHelper.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

```
204 ) external returns (uint256, uint256, AddParams memory,
      ↪ LiquidityLoan[] memory) {
```

CVF-125 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “IERC20“.

```
301 address token,
```

CVF-126 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
304 address poolManager,
```

CVF-127 INFO

- **Category** Suboptimal
- **Source** MarginPositionHelper.sol

Recommendation This could be simplified using the “max” function.

```
340 fillerOutput = order.minOutput > fillerOutput ? order.minOutput :
      ↪ fillerOutput;
```

CVF-128 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The value “1e18” should be a named constant.

```
350 require(param.reducePercentage != 0 && param.reducePercentage <=1e18
    ↪ , "incorrectReduce");
```

```
370 = Utils.getAmountsRequired(vars.repayInfo, 1e18, vars.
    ↪ tickDiscretization, vars.curTick, vars.curSqrtPriceX96);
```

```
456 Utils.getAmountsRequired(vars.repayInfo, 1e18, vars.
    ↪ tickDiscretization, vars.curTick, vars.curSqrtPriceX96);
```

```
546 Utils.getAmountsRequired(vars.repayInfo, 1e18, vars.
    ↪ tickDiscretization, vars.curTick, vars.curSqrtPriceX96);
```

```
686 Utils.getAmountsRequired(borrowInfo, 1e18, tickDiscretization,
    ↪ curTick, curSqrtPriceX96);
```

CVF-129 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
380 } else if (param.executionOption == 2) {} else if (param.
    ↪ executionOption == 3) {
```

CVF-130 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
598 address poolManager,
```

CVF-131 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “Executioner”.

```
624 address executioner,
```

CVF-132 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
625 address poolManager
```

CVF-133 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
657 address poolManager
```

CVF-134 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “Executioner”.

```
680 address executioner,
```

CVF-135 INFO

- **Category** Bad datatype
- **Source** MarginPositionHelper.sol

Recommendation The type for this argument should be “PoolManager”.

```
681 address poolManager,
```

CVF-136 INFO

- **Category** Procedural
- **Source** MarginPositionHelper.sol

Recommendation This comment shouldn’t be here.

```
735 // in the future, for decentralization, we can do 1) dutch auction
       ↪ or 2) paying with UPNL for force closes
```

CVF-137 INFO

- **Category** Procedural
- **Source** MarginFacility.sol

Description We didn’t review these files.

```
5  import "./types.sol";
```

```
9  import {ERC20} from "./ERC20.sol";
```

```
13 import {SafeTransferLib} from "./utils/SafeTransferLib.sol";
```

CVF-138 INFO

- **Category** Procedural
- **Source** MarginFacility.sol

Description It is unclear, what versions of these files are used.

Recommendation Consider importing these files directly from the Uniswap repository.

```
7  import {IUniswapV3Pool} from "./uniswap/interfaces/IUniswapV3Pool.
       ↪ sol";
   import {FullMath} from "./uniswap/FullMath.sol";

10 import {TransferHelper} from "./uniswap/TransferHelper.sol";
```

CVF-139 INFO

- **Category** Bad naming
- **Source** MarginFacility.sol

Recommendation Events are usually named via nouns, such as “MarginPositionIncrease” or “MarginPositionReduction”.

```
29 event MarginPositionIncreased(

42 event MarginPositionReduced(address indexed pool,bool indexed
       ↪ positionIsToken0,bool marginInPosToken, address indexed trader
       ↪ ,address filler,uint256 reduceAmount,uint256 premiumPaid,
       ↪ uint256 feePaid, int256 PnL);
   event ForceClosed(address indexed pool,bool indexed positionIsToken0
       ↪ ,bool marginInPosToken,uint margin, address indexed trader,
       ↪ uint256 forcedClosedAmount,uint256 fillerPayAmount,uint256
       ↪ rangeCondition);

45 event OrderAdded(

58 event OrderCanceled(address indexed pool, bool indexed
       ↪ positionIsToken0, address indexed trader, bool isAdd);
```

CVF-140 INFO

- **Category** Bad datatype
- **Source** MarginFacility.sol

Recommendation The type for the “pool” parameters should be more specific.

```
30     address indexed pool,
```

```
42 event MarginPositionReduced(address indexed pool,bool indexed
   ↪ positionIsToken0,bool marginInPosToken, address indexed trader
   ↪ ,address filler,uint256 reduceAmount,uint256 premiumPaid,
   ↪ uint256 feePaid, int256 PnL);
   event ForceClosed(address indexed pool,bool indexed positionIsToken0
   ↪ ,bool marginInPosToken,uint margin, address indexed trader,
   ↪ uint256 forcedClosedAmount,uint256 fillerPayAmount,uint256
   ↪ rangeCondition);
```

```
46     address indexed pool,
```

```
58 event OrderCanceled(address indexed pool, bool indexed
   ↪ positionIsToken0, address indexed trader, bool isAdd);
```

CVF-141 INFO

- **Category** Unclear behavior
- **Source** MarginFacility.sol

Description This function should emit some event.

```
60 function addFiller(address filler) external onlyOwner{
```

CVF-142 INFO

- **Category** Bad datatype
- **Source** MarginFacility.sol

Recommendation The type for the “pool” arguments should be more specific.

```
64  function checkPositionExists(address pool, address borrower, bool
    ↪ borrowedToken1)

73  function getBorrowInfo(address pool, address borrower, bool
    ↪ borrowedToken1)

83  function getPosition(address pool, address trader, bool
    ↪ positionIsToken0)

91  function getLastRepayTime(address pool, address trader, bool
    ↪ positionIsToken0)

103     address pool,

112 function _updateBorrowInfo(address pool, address borrower, bool
    ↪ borrowedToken1, LiquidityLoan[] memory borrowInfo)

126 function _updateLastPremiumPaymentTime(address pool, address
    ↪ borrower, bool borrowedToken1) internal override {

422     address pool,

443 function cancelOrder(address pool, bool positionIsToken0, bool isAdd
    ↪ ) external nonReentrant {
```

CVF-143 INFO

- **Category** Procedural
- **Source** MarginFacility.sol

Recommendation Outer brackets are redundant.

```
70  return (positions[getPositionId(pool, borrower, borrowedToken1)].
    ↪ totalPosition > 0);
```

CVF-144 INFO

- **Category** Bad datatype
- **Source** MarginFacility.sol

Recommendation The value “1e18” should be a named constant.

```
264 vars.repayInfo = param.reducePercentage == 1e18
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting