Report

v. 1.0

Customer

Equation

Smart Contract Audit

Equation DAO

25th October 2023

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. FIXED . . . 11
CVF-9. FIXED . . . 11
CVF-10. FIXED . . . 12
CVF-13. FIXED . . . 12
CVF-17. FIXED . . . 13
CVF-26. FIXED . . . 13

7 Moderate Issues **14**
CVF-31. FIXED . . . 14
CVF-5. INFO . . . 15
CVF-6. INFO . . . 15
CVF-15. INFO . . . 16
CVF-16. INFO . . . 17
CVF-21. INFO . . . 18
CVF-24. INFO . . . 18
CVF-25. INFO . . . 19
CVF-27. FIXED . . . 19
CVF-28. FIXED . . . 19
CVF-29. INFO . . . 20
CVF-30. INFO . . . 20
CVF-32. INFO . . . 21
CVF-33. INFO . . . 21
CVF-34. INFO . . . 21
CVF-35. INFO . . . 22
CVF-36. INFO . . . 23
CVF-38. INFO . . . 23
CVF-40. INFO . . . 24
CVF-41. INFO . . . 24
CVF-42. INFO . . . 24
CVF-43. FIXED . . . 25
CVF-44. INFO . . . 25
CVF-45. INFO . . . 26
CVF-46. INFO . . . 26

CVF-47. FIXED 26
CVF-48. INFO 27

8 Minor Issues **28**
CVF-2. FIXED 28
CVF-3. FIXED 28
CVF-4. FIXED 28
CVF-7. FIXED 29
CVF-8. FIXED 29
CVF-11. INFO 29
CVF-12. INFO 30
CVF-14. INFO 30
CVF-19. INFO 31
CVF-22. FIXED 31
CVF-23. INFO 31
CVF-37. INFO 32
CVF-39. INFO 32
CVF-49. FIXED 32
CVF-50. INFO 33
CVF-51. FIXED 33
CVF-52. INFO 34
CVF-53. FIXED 35
CVF-54. FIXED 35
CVF-55. FIXED 36
CVF-56. INFO 36
CVF-57. INFO 36
CVF-58. INFO 37
CVF-59. INFO 37
CVF-60. FIXED 37
CVF-61. INFO 38
CVF-62. FIXED 38
CVF-63. FIXED 38
CVF-64. INFO 39
CVF-65. INFO 39
CVF-66. FIXED 40
CVF-67. FIXED 40
CVF-68. INFO 41
CVF-69. FIXED 41
CVF-70. FIXED 42
CVF-71. FIXED 42
CVF-72. INFO 43
CVF-73. INFO 44
CVF-74. FIXED 45
CVF-75. FIXED 46
CVF-76. FIXED 47
CVF-77. INFO 48

CVF-78. FIXED 48
CVF-79. INFO 48
CVF-80. INFO 49
CVF-81. INFO 49
CVF-82. INFO 49
CVF-83. INFO 50
CVF-84. INFO 50
CVF-85. INFO 50
CVF-86. INFO 51
CVF-87. INFO 51
CVF-88. INFO 51
CVF-89. INFO 52
CVF-90. FIXED 52
CVF-91. INFO 52
CVF-92. INFO 53
CVF-93. FIXED 53
CVF-94. INFO 53
CVF-95. FIXED 54
CVF-96. FIXED 54
CVF-97. FIXED 54
CVF-98. INFO 55
CVF-99. FIXED 56
CVF-100. INFO 56
CVF-101. INFO 56
CVF-102. INFO 57
CVF-103. INFO 57
CVF-104. FIXED 57
CVF-105. INFO 58
CVF-106. INFO 58
CVF-107. INFO 58
CVF-108. FIXED 59
CVF-109. FIXED 59
CVF-110. FIXED 59
CVF-111. FIXED 60
CVF-112. INFO 60
CVF-113. INFO 60
CVF-114. INFO 61
CVF-115. INFO 62
CVF-116. FIXED 62
CVF-117. FIXED 62
CVF-118. INFO 63
CVF-119. FIXED 63
CVF-120. INFO 63
CVF-121. FIXED 64
CVF-122. FIXED 64
CVF-123. INFO 65

CVF-124. INFO 65
CVF-125. FIXED 66
CVF-126. INFO 67
CVF-127. INFO 68
CVF-128. INFO 68
CVF-129. INFO 69
CVF-130. INFO 70
CVF-131. INFO 70
CVF-132. INFO 71
CVF-133. FIXED 71
CVF-134. FIXED 71
CVF-135. INFO 71
CVF-136. INFO 72
CVF-137. INFO 72
CVF-138. INFO 72
CVF-139. INFO 73
CVF-140. INFO 73
CVF-141. INFO 73
CVF-142. INFO 74
CVF-143. INFO 74
CVF-144. INFO 74
CVF-145. INFO 75
CVF-146. INFO 75

1 Changelog

#	Date	Author	Description
0.1	25.10.23	A. Zveryanskaya	Initial Draft
0.2	25.10.23	A. Zveryanskaya	Minor revision
1.0	25.10.23	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.

In traditional Tokenomics models, the complex interests between community users, the founding team, and early investors often lead to disagreements and divergent consensus during the operation and governance of the protocol. This greatly hampers the sustainable development of the protocol.
Equation adopts an innovative hybrid Tokenomics model that combines Fungible Tokens (FT) and Non-Fungible Tokens (NFT). This hybrid model establishes exclusive incentive channels for different roles in the ecosystem, aiming to unify the long-term development goals of Equation while safeguarding the interests of liquidity providers and community users.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

core/		
Configurable.sol	Pool.sol	PoolFactory.sol
core/interfaces/		
IConfigurable.sol	IPool.sol	IPoolErrors.sol
IPoolFactory.sol	IPoolLiquidityPosition.sol	IPoolPosition.sol
farming/		
RewardFarm.sol		
farming/interfaces/		
IRewardFarm.sol	IRewardFarmCallback.sol	
libraries/		
Constants.sol	FundingRateUtil.sol	LiquidityPositionUtil.sol
Math.sol	PoolUtil.sol	PositionUtil.sol
PriceUtil.sol		
oracle/		
PriceFeed.sol		
oracle/interfaces/		
IPriceFeed.sol		
plugins/		
OrderBook.sol	PluginManager.sol	PositionRouter.sol
RewardCollector.sol	Router.sol	
plugins/interfaces/		
IOrderBook.sol	IPluginManager.sol	IPositionRouter.sol
staking/		
FeeDistributor.sol		
staking/interfaces/		
IFeeDistributor.sol	IFeeDistributor Callback.sol	
types/		
Bitmap.sol	Side.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.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 6 major, and a few less important issues. All identified Major issues have been fixed.

	Info	Fixed
Major	0	6
Moderate	22	5
Minor	69	42

Fixed 53 out of 144 issues

6 Major Issues

CVF-1. FIXED

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

Description This operation may hide errors and may make it more risky introducing additions sides in the future.

```
11 function normalize(Side self) pure returns (Side) {
```

CVF-9. FIXED

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

Description According to the documentation comment, the “true” return value means that the cancellation was successful, while here such value is returned when the index of a non-existing request was passed as an argument, and no cancellation actually happened.

Recommendation Consider reverting in such a case, or returning “false”, or changing the documentation comment.

```
127 if (request.account == address(0)) return true;
227 if (request.account == address(0)) return true;
328 if (request.account == address(0)) return true;
423 if (request.account == address(0)) return true;
517 if (request.account == address(0)) return true;
624 if (request.account == address(0)) return true;
741 if (request.account == address(0)) return true;
```

CVF-10. FIXED

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

Description According to the documentation comment, the “true” return value means that the execution was successful, while here such value is returned when the index of a non-existing request was passed as an argument, and no execution actually happened.

Recommendation Consider reverting in such a case, or returning “false”, or changing the documentation comment.

```
149 if (request.account == address(0)) return true;
247 if (request.account == address(0)) return true;
350 if (request.account == address(0)) return true;
445 if (request.account == address(0)) return true;
537 if (request.account == address(0)) return true;
646 if (request.account == address(0)) return true;
761 if (request.account == address(0)) return true;
```

CVF-13. FIXED

- **Category** Flaw
- **Source** PriceFeed.sol

Recommendation This should be checked by the caller, as this function doesn’t know where this value came from.

```
234 if (address(_aggregator) == address(0)) revert
      ↪ ReferencePriceFeedNotSet();
```

CVF-17. FIXED

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

Recommendation This should be surrounded by an “unchecked” block to property support the type(int256).min value.

```
35 return unrealizedPnL >= 0 ? 0 : uint256(-unrealizedPnL);
```

CVF-26. FIXED

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

Recommendation Should be “-int256(cumulativePremiumRateX96)” to prevent possible underflow.

```
182 : -int256(Math.ceilDiv(uint256(int256(-cumulativePremiumRateX96)),
      ↪ Constants.PREMIUM_RATE_AVG_DENOMINATOR));
```

7 Moderate Issues

CVF-31. FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Description While the “mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the numerator is a power of two, the multiplication could be replaced with a shift. When the denominator is a compile-time constant, the modular reciprocal of the denominator could be precomputed.

Recommendation Consider implementing such optimized functions.

880
```
        Math.mulDiv(
            _margin - _liquidationExecutionFee,
            tokenConfig.maxRiskRatePerLiquidityPosition,
            Constants.BASIS_POINTS_DIVISOR
        ) <=
```

890
```
        Math.mulDiv(
            _margin - _liquidationExecutionFee,
            tokenConfig.maxRiskRatePerLiquidityPosition,
            Constants.BASIS_POINTS_DIVISOR
        ) >
```

1143
```
Math.mulDiv(liquidityFee, Constants.Q64, _positionCache.liquidity);
```

1195
```
    Math.mulDivUp(
        tokenFeeRateConfig.tradingFeeRate,
        tokenFeeRateConfig.referralDiscountRate,
        Constants.BASIS_POINTS_DIVISOR
    )
```

1263
```
    .mulDiv(uint256(insufficientFundingFee), Constants.Q96,
       ↪ oppositeSize)
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** FeeDistributor.sol

Description While the “mulDiv” function is very efficient in generic case, for particular cases better approaches to exist. For example, when the numerator or the denominator is a power of two, multiplication or division could be replaced with a shift.

Recommendation Consider implementing such optimized functions.

Client Comment *The result of “x * y” may exceed “type(uint256).max”.*

115
```
perShareGrowthX64 += Math.mulDiv(equFeeAmount, Constants.Q64,
    ↪ totalStakedWithMultiplier).toUint160();
architectPerShareGrowthX64 += Math.mulDiv(architectFeeAmount,
    ↪ Constants.Q64, mintedArchitects).toUint160();
```

321
```
    amount = Math.mulDiv(
        perShareGrowthX64 - stakeInfoCache.perShareGrowthX64,
        stakeInfoCache.amount * stakeInfoCache.multiplier,
        Constants.Q64
    );
```

CVF-6. INFO

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

Description Overflow protection is supposed to be used as a second like of defence, in case some business-level constraints were violated.

Recommendation Consider using checked math even when overflow should never happen from business point of view.

Client Comment *There won’t be an overflow here, and comments have already been added.*

266
```
// Because the total amount of EQU issued is 10 million, it will
    ↪ never overflow here.
totalStakedAmount += stakeInfoCache.amount;
totalStakedWithMultiplierDelta += stakeInfoCache.amount *
    ↪ stakeInfoCache.multiplier;
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** PoolUtil.sol

Description Internally, three external calls are made to the same contract.

Recommendation Consider refactoring to merge these calls into one call.

Client Comment *These functions are defined as private.*

```
16 _changeTokenConfig(_tokenConfig, _poolFactory, _token);
18 _changeTokenFeeRateConfig(_tokenFeeRateConfig, _poolFactory, _token)
       ↪ ;
20 _changeTokenPriceConfig(_priceState, _poolFactory, _token);
```

CVF-16. INFO

- **Category** Suboptimal
- **Source** PositionUtil.sol

Description While the “mulDiv” function is very efficient in generic case, for specific cases better approaches do exists. For example, when the denominator is a power of two, the division could be replaced with a shift, and when the denominator is a compile-time constant, the modular reciprocal of denominator could be precomputed.

Recommendation Consider using such optimizations.

Client Comment *The result of “x * y” may exceed “type(uint256).max”.*

```
48  liquidity = Math.mulDivUp(_size, _priceX96, Constants.Q96).toUint128
        ↪ ();
```

```
71              unrealizedPnL = -int256(Math.mulDivUp(_size,
                    ↪ _entryPriceX96 - _priceX96, Constants.Q96));
            else unrealizedPnL = int256(Math.mulDiv(_size, _priceX96 -
                ↪ _entryPriceX96, Constants.Q96));
```

```
75              unrealizedPnL = -int256(Math.mulDivUp(_size, _priceX96 -
                    ↪  _entryPriceX96, Constants.Q96));
            else unrealizedPnL = int256(Math.mulDiv(_size,
                ↪ _entryPriceX96 - _priceX96, Constants.Q96));
```

```
135 if (deltaX96 >= 0) fundingFee = Math.mulDiv(uint192(deltaX96),
        ↪ _positionSize, Constants.Q96).toInt256();
    else fundingFee = -Math.mulDivUp(uint192(-deltaX96), _positionSize,
        ↪ Constants.Q96).toInt256();
```

```
161     maintenanceMargin = Math.mulDivUp(
            _size,
            uint256(_entryPriceX96) * _liquidationFeeRate + uint256(
                ↪ _indexPriceX96) * _tradingFeeRate,
            Constants.BASIS_POINTS_DIVISOR * Constants.Q96
        );
```

CVF-21. INFO

- **Category** Suboptimal
- **Source** PriceUtil.sol

Description Casting “this” to a particular interface inside a library is weird. Also, making an external call here is suboptimal.

Recommendation Consider refactoring. For example, consider passing a function reference as an argument into the library call.

Client Comment *Better readability compared to passing a function reference.*

```
88 IPool(address(this)).changePriceVertex(
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** PriceUtil.sol

Description While the “mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the denominator is a power of 2, the division could be replaced with a shift.

Recommendation Consider implementing such an optimized function.

Client Comment *The result of “x * y” may exceed “type(uint256).max”.*

```
267 (uint256 tradePriceX96Down, uint256 tradePriceX96Up) = Math.mulDiv2(
```

```
272     Constants.Q96 << 1
```

```
313     ? Math.mulDivUp(_indexPriceX96, premiumRateAfterX96, Constants.
          ↪ Q96).toUint160()
        : Math.mulDiv(_indexPriceX96, premiumRateAfterX96, Constants.Q96
          ↪ ).toUint160();
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** FundingRateUtil.sol

Description While the “mulDiv” function is very efficient in generic case, for specific cases better approaches do exists. For example, when the numerator or the denominator is a power of 2, it is possible to replace multiplication or the division by shift.

Recommendation Consider implementing such optimized functions.

Client Comment *The result of “x * y” may exceed “type(uint256).max”.*

```
109     .mulDivUp(_indexPriceX96, clampedFundingRateDeltaAbsX96,
           ↪ Constants.Q96)
```

```
125             Math.mulDiv(paidSize, uint192(
                   ↪ paidFundingRateGrowthDeltaX96), Constants.Q96)
```

```
145 return int256(Math.mulDivUp(_maxFundingRate, Constants.Q96,
       ↪ Constants.BASIS_POINTS_DIVISOR));
```

```
195 int256 interestRateX96 = int256(Math.mulDivUp(_interestRate,
       ↪ Constants.Q96, Constants.BASIS_POINTS_DIVISOR));
```

CVF-27. FIXED

- **Category** Suboptimal
- **Source** RewardFarm.sol

Recommendation Bitwise AND is redundant here, as conversion to “uint8” does the same.

```
418 return uint8(_poolIndex & (~uint8(0)));
```

CVF-28. FIXED

- **Category** Suboptimal
- **Source** Configurable.sol

Description Checking this on every loop iteration is redundant.

Recommendation Consider checking only for the last vertex.

```
158 _newCfg.vertices[i].balanceRate > Constants.BASIS_POINTS_DIVISOR ||
    _newCfg.vertices[i].premiumRate > Constants.BASIS_POINTS_DIVISOR
```

CVF-29. INFO

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

Description The validation is hidden inside a function. Relying on constraints enforced in distant parts of code is a bad practice, as it creates hidden relationships between code parts.

Recommendation Consider using checked subtraction.

Client Comment *For gas-saving purposes, and there are also comments added here.*

```
189 // never underflow because of the validation above
190 marginAfter -= positionUnrealizedLoss;
```

CVF-30. INFO

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

Description It is unclear, whether the comment describes a desired behavior or a known problem.

Recommendation If this is a known problem, then consider surrounding with an "unchecked" block. If this is a desired behavior, consider checking explicitly and reverting with a meaningful error.

Client Comment *This is the desired behavior, and it has already been documented through comment.*

```
247 // If marginDelta is equal to type(int128).min, it will revert here
    if (marginAfter < uint128(-_marginDelta)) revert InsufficientMargin
       ↪ ();
```

CVF-32. INFO

- **Category** Suboptimal
- **Source** PoolFactory.sol

Description Deploying a separate contract for every pool is quite expensive.

Recommendation Consider deploying a minimalistic proxy for a shared implementation instead.

Client Comment *This can save gas when users interact with the contract.*

```
102 _pool = IPool(Create2.deploy(0, keccak256(abi.encode(_token, usd)),
        ↪ creationCode));
```

CVF-33. INFO

- **Category** Suboptimal
- **Source** FeeDistributor.sol

Description This function overwrites multipliers for a sequence of periods starting at the index zero, but leaves the remaining periods untouched. This behavior seems weird, as in order to set the multiplier for a period, one needs to set multipliers for all the preceding periods.

Recommendation Consider addressing this. For example, consider accepting the starting index of the periods to set the multipliers for.

Client Comment *The function is an incremental update method.*

```
95 function setLockupRewardMultipliers(
```

CVF-34. INFO

- **Category** Procedural
- **Source** PluginManager.sol

Recommendation This contract inherits the “Governable” contract that we didn’t review.

Client Comment *This contract is very simple and doesn’t require an audit.*

```
7 abstract contract PluginManager is IPluginManager, Governable {
```

CVF-35. INFO

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

Description By manipulating the transaction gas limit it is possible to make the execute call fail, and then probably the cancel call to succeed.

Recommendation Consider guaranteeing certain gas amount for the execute call.

Client Comment *GasLimit has been set in each 'executeXXX'.*

182
```
try this.executeOpenLiquidityPosition(index, _executionFeeReceiver)
    ↪ returns (bool _executed) {
    if (!_executed) break;
} catch {
    try this.cancelOpenLiquidityPosition(index,
        ↪ _executionFeeReceiver) returns (bool _cancelled) {
        if (!_cancelled) break;
    } catch {}
}
```

271
```
try this.executeCloseLiquidityPosition(index, _executionFeeReceiver)
    ↪  returns (bool _executed) {
    if (!_executed) break;
} catch {
    try this.cancelCloseLiquidityPosition(index,
        ↪ _executionFeeReceiver) returns (bool _cancelled) {
        if (!_cancelled) break;
    } catch {}
}
```

381
```
try this.executeAdjustLiquidityPositionMargin(index,
    ↪ _executionFeeReceiver) returns (bool _executed) {
    if (!_executed) break;
} catch {
    try this.cancelAdjustLiquidityPositionMargin(index,
        ↪ _executionFeeReceiver) returns (bool _cancelled) {
        if (!_cancelled) break;
    } catch {}
}
```

(475, 566, 681, 795)

CVF-36. INFO

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

Description Overflow is possible when converting types.

Recommendation Consider using safe conversion.

Client Comment *There won't be an overflow here, and comments have already been added.*

```
71      unrealizedPnL = -int256(Math.mulDivUp(_size, _entryPriceX96 -
            ↪ _priceX96, Constants.Q96));
    else unrealizedPnL = int256(Math.mulDiv(_size, _priceX96 -
        ↪ _entryPriceX96, Constants.Q96));
```

```
75      unrealizedPnL = -int256(Math.mulDivUp(_size, _priceX96 -
            ↪ _entryPriceX96, Constants.Q96));
    else unrealizedPnL = int256(Math.mulDiv(_size, _entryPriceX96 -
        ↪ _priceX96, Constants.Q96));
```

CVF-38. INFO

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

Description Overflow is possible when converting to "uint128".

Recommendation Consider using safe conversion.

Client Comment *The type of _sizeDelta here is uint128, and since _sizeDelta is greater than netSizeAfter, it can be safely converted.*

```
155 uint128 sizeUsed = _sizeDelta > netSizeAfter ? uint128(netSizeAfter)
        ↪  : _sizeDelta;
```

CVF-40. INFO

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

Description An unchecked underflow is possible here.

Recommendation Consider using safe subtraction.

Client Comment *There will be no overflow, as simulateMove already restricts sizeUsed from exceeding sizeLeft.*

```
171 unchecked { step.sizeLeft -= sizeUsed; }
```

CVF-41. INFO

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

Description Unchecked underflow is possible here.

Client Comment *Due to the processing at line 38, the range of timeDelta is [0, 3600], so there won't be an overflow.*

```
45 uint16 timeDelta = uint16(_currentTimestamp - lastSamplingTime);
```

CVF-42. INFO

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

Description Overflow when converting type is possible here.

Client Comment *Due to the processing at line 38, the range of timeDelta is [0, 3600], so there won't be an overflow.*

```
45 uint16 timeDelta = uint16(_currentTimestamp - lastSamplingTime);
```

CVF-43. FIXED

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

Description This code is very confusing. The fields “SidePosition.long” and “SidePosition.short” are calculated using the “mulDiv” function so they should be numbers, but at the other hand, they are used in bitwise expressions, so they should be bit maps.

Recommendation Consider explaining or refactoring.

```
171 SidePosition memory sidePositionCache = sidePosition;

174 uint128 positionAfter = Math.mulDiv(_sizeAfter, _entryPriceAfterX96,
       ↪  Constants.Q96).toUint128();
    uint128 beforeMasked = sidePositionCache.long | sidePositionCache.
       ↪ short;

179     afterMasked = positionAfter | sidePositionCache.short;

181     sidePosition.long = positionAfter;

184     afterMasked = positionAfter | sidePositionCache.long;

186     sidePosition.short = positionAfter;
```

CVF-44. INFO

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

Description Underflow and overflow are possible here.

Recommendation Consider using checked math and safe conversions.

Client Comment *There won’t be any overflow, as the increase and decrease of liquidity are already guaranteed by the Pool.*

```
705 if (_liquidityDelta < 0) liquidityAfter = uint128(_liquidity -
       ↪ uint256(-_liquidityDelta));
    else liquidityAfter = uint128(_liquidity + uint256(_liquidityDelta))
       ↪ ;
```

CVF-45. INFO

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

Description Underflow is possible here.

Recommendation Consider using checked subtraction.

Client Comment *There won't be any overflow, as _rewardGrowthX64 is a snapshot of _globalRewardGrowthX64, and _globalRewardGrowthX64 is an accumulated value. _rewardGrowthX64 will not exceed _globalRewardGrowthX64.*

```
716 unchecked { rewardDebt = Math.mulDiv(_globalRewardGrowthX64 -
    ↪ _rewardGrowthX64, _liquidity, Constants.Q64); }
```

CVF-46. INFO

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

Description Underflow is possible here.

Recommendation Consider using checked subtraction.

Client Comment *There won't be any underflow here.*

```
194 globalLiquidityPosition.liquidity = globalPositionCache.liquidity -
    ↪ positionCache.liquidity;
```

CVF-47. FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Description If "_amount" is less than the balance change, the excess amount is silently consumed.

Recommendation Consider refunding it or requiring exact equality here. Alternatively, consider increased "usdBalance" by "_amount" rather then setting to the actual balance.

```
843 if (balanceAfter - usdBalance < _amount) revert InsufficientBalance(
    ↪ usdBalance, _amount);
```

CVF-48. INFO

- **Category** Suboptimal
- **Source** PoolFactory.sol

Description This line is very gas consuming. It reads the already stored part of the creation code into memory, concatenates in the memory and then write the concatenated creation code back into the storage.

Recommendation It would be more efficient to just append in the storage.

Client Comment *This function will only execute twice, and the second time requires reading all data to calculate the hash.*

```
46 creationCode = bytes.concat(creationCode, _creationCodePart);
```

8 Minor Issues

CVF-2. FIXED

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

Description This function is inconsistent with the “normalize” function.

Recommendation Consider treating all non-long sides as short.

```
19 function isShort(Side self) pure returns (bool) {
```

CVF-3. FIXED

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

Description This function is inconsistent with the “normalize“ function.

Recommendation Consider treating two non-long sides as equal even if their underlying values differ.

```
23 function eq(Side self, Side other) pure returns (bool) {
```

CVF-4. FIXED

- **Category** Suboptimal
- **Source** Bitmap.sol

Recommendation This function could be significantly optimized like this: https://github.com/vectorized/solady/blob/main/src/utils/LibBit.sol#L47-L68

```
33 function leastSignificantBit(uint256 x) pure returns (uint8 r) {
```

CVF-7. FIXED

- **Category** Suboptimal
- **Source** FeeDistributor.sol

Description Checking this condition on every loop iteration is suboptimal.

Recommendation Consider checking before the loop, and then implementing two different loops, one for UniswapV3 positions, and another for normal stakes.

300
```
if (_isUniswapV3Pos) {
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** Router.sol

Description This could potentially check the same owner several times.

Recommendation Consider not checking an owner in case it is the same as the owner of the previous token. This would allow saving gas by sorting tokens by owner address.

198
```
for (uint256 i; i < _referralTokens.length; ++i) _onlyPluginApproved
    ↪ (EFC.ownerOf(_referralTokens[i]));
```

239
```
for (uint256 i; i < _tokenIDs.length; ++i) _onlyPluginApproved(EFC.
    ↪ ownerOf(_tokenIDs[i]));
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** RewardCollector.sol

Description The “ownerOf” call is performed twice.

Recommendation Consider performing once and caching the result.

Client Comment *Since the normal scenario is typically “owner” caching the result using a variable would increase gas consumption in normal scenarios.*

77
```
if (EFC.ownerOf(_referralTokens[i]) != caller)
    revert InvalidCaller(caller, EFC.ownerOf(_referralTokens[i]));
```

CVF-12. INFO

- **Category** Suboptimal
- **Source** PriceFeed.sol

Description Each of these functions modify certain property of a token configuration.

Recommendation Consider implementing a way to modify several properties at once atomically.

Client Comment *Because configuration functions are important low-frequency operations, to ensure that there are no errors in modifying associated data, we choose to separate them.*

```
156 function setRefPriceFeeds(IERC20 _token, IChainLinkAggregator
    ↪ _priceFeed) external override onlyGov {
```

```
166 function setRefHeartbeatDuration(IERC20 _token, uint32 _duration)
    ↪ external override onlyGov {
```

```
181 function setMaxCumulativeDeltaDiffs(IERC20 _token, uint64
    ↪ _maxCumulativeDeltaDiff) external override onlyGov {
```

CVF-14. INFO

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

Recommendation In some cases a hardcoded value “7” is used instead of this constant.

Client Comment *The current compiler version doesn’t support “PriceVertex[Constants.VERTEX_NUM]”, so hardcoding is the only option.*

```
15 uint8 internal constant VERTEX_NUM = 7;
```

CVF-19. INFO

- **Category** Readability
- **Source** LiquidityPositionUtil.sol

Description Here an argument is used as a local variable.

Recommendation Consider using a separate local variable instead.

Client Comment *Using the same variable can save gas.*

```
163 _sizeDelta -= sizeUsed;
```

CVF-22. FIXED

- **Category** Readability
- **Source** PriceUtil.sol

Description Using a signed type for "i" here makes the code cumbersome.

Recommendation Consider refactoring for example like this: for (uint8 i = _priceStateCache.currentVertexIndex; true; i -= 1) { ...; if (i == 0) break; }

```
191 for (int8 i = int8(_priceStateCache.currentVertexIndex); i >= 0 &&
    ↪ step.sizeLeft > 0; --i) {
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** PriceUtil.sol

Description One one of the two returned values is actually used.

Recommendation Consider calculating only the needed value.

Client Comment *Compared to separately calculating the needed values, this has better readability.*

```
267 (uint256 tradePriceX96Down, uint256 tradePriceX96Up) = Math.mulDiv2(
```

CVF-37. INFO

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

Description Unchecked underflow is possible here.

Recommendation Consider using safe subtraction.

Client Comment *There won't be any underflow here.*

```
111 uint128 liquidityDelta = _globalLiquidity - _metricsCache.liquidity;
```

CVF-39. INFO

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

Description Unchecked underflow is possible here.

Recommendation Consider using safe subtraction.

Client Comment *There won't be any underflow here.*

```
260 _riskBufferFund.liquidity -= _liquidityDelta;
```

CVF-49. FIXED

- **Category** Procedural
- **Source** Side.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version. Also relevant for:Bitmap.sol, IFeeDistributor.sol, IFeeDistributorCallback.sol, IPluginManager.sol,IOrderBook.sol, IPositionRouter.sol, PluginManager.sol, IPriceFeed.sol, Constants.sol, PoolUtil.sol, Math.sol, PositionUtil.sol, LiquidityPositionUtil.sol, PriceUtil.sol, FundingRateUtil.sol, IRewardFarmCallback.sol, IRewardFarm.sol, IConfigurable.sol, IPoolErrors.sol, IPoolPosition.sol, IPool.sol, IPoolFactory.sol, Configurable.sol.

```
2 pragma solidity ^0.8.21;
```

CVF-50. INFO

- **Category** Suboptimal
- **Source** Bitmap.sol

Recommendation This could be optimized as: self » startInclusive « startInclusive

Client Comment *The current code is clear and doesn't need any changes.*

```
22 uint256 mask = ~uint256(0) << startInclusive;
   uint256 masked = Bitmap.unwrap(self) & mask;
```

CVF-51. FIXED

- **Category** Procedural
- **Source** IFeeDistributor.sol

Description The parameter names are different, while description are the same.

Recommendation Consider using the same names.

```
10 /// @param stakeID Index of EQU tokens staking information
```

```
26 /// @param id Index of EQU tokens staking information
```

```
42 /// @param id Index of EQU tokens staking information
```

CVF-52. INFO

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

Recommendation Events are usually named via nouns, such as “Stake”, “V3PosStake”, “Unstake” etc.

```
13 event Staked(address indexed sender, address indexed account,
   ↪ uint256 stakeID, uint256 amount, uint16 period);

21 event V3PosStaked(address indexed sender, address indexed account,
   ↪ uint256 stakeID, uint256 amount, uint16 period);

29 event Unstaked(address indexed owner, address indexed receiver,
   ↪ uint256 id, uint256 amount0, uint256 amount1);

37 event V3PosUnstaked(address indexed owner, address indexed receiver,
   ↪  uint256 id, uint256 amount0, uint256 amount1);

44 event Collected(address indexed owner, address indexed receiver,
   ↪ uint256 id, uint256 amount);

51 event V3PosCollected(address indexed owner, address indexed receiver
   ↪ , uint256 id, uint256 amount);

57 event ArchitectCollected(address indexed receiver, uint256 tokenID,
   ↪ uint256 amount);

65 event FeeDeposited(

76 event LockupRewardMultipliersSet(uint16[] periods, uint16[]
   ↪ multipliers);
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** IFeeDistributor.sol

Recommendation The ID parameters should be indexed.

```
13 event Staked(address indexed sender, address indexed account,
   ↪ uint256 stakeID, uint256 amount, uint16 period);

21 event V3PosStaked(address indexed sender, address indexed account,
   ↪ uint256 stakeID, uint256 amount, uint16 period);

29 event Unstaked(address indexed owner, address indexed receiver,
   ↪ uint256 id, uint256 amount0, uint256 amount1);

37 event V3PosUnstaked(address indexed owner, address indexed receiver,
   ↪  uint256 id, uint256 amount0, uint256 amount1);

44 event Collected(address indexed owner, address indexed receiver,
   ↪ uint256 id, uint256 amount);

51 event V3PosCollected(address indexed owner, address indexed receiver
   ↪ , uint256 id, uint256 amount);

57 event ArchitectCollected(address indexed receiver, uint256 tokenID,
   ↪ uint256 amount);
```

CVF-54. FIXED

- **Category** Procedural
- **Source** IFeeDistributor.sol

Description The parameter names are different, while description are the same.

Recommendation Consider using the same names.

```
18 /// @param stakeID Index of Uniswap V3 positions NFTs staking
   ↪ information

34 /// @param id Index of Uniswap V3 positions NFTs staking information

49 /// @param id Index of Uniswap V3 positions NFTs staking information
```

CVF-55. FIXED

- **Category** Suboptimal
- **Source** IFeeDistributor.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
88 error InvalidLockupPeriod();
```

```
96 error InvalidUniswapV3PositionNFT();
```

CVF-56. INFO

- **Category** Procedural
- **Source** FeeDistributor.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for: Router.sol, PositionRouter.sol, RewardCollector.sol, PriceFeed.sol, RewardFarm.sol, Pool.sol, PoolFactory.sol.

Client Comment *Using a compiler version that has been tested can help ensure there are no hidden security issues.*

```
2 pragma solidity =0.8.21;
```

CVF-57. INFO

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

Recommendation The type of this variable should be more specific.

Client Comment *This address is only used for calculating the Pool address and, therefore, doesn't need to be specified as a more specific type.*

```
30 address private immutable v3PoolFactory;
```

CVF-58. INFO

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

Description The variable name is confusing, as it is unclear which token is token 0.

Recommendation Consider using a more descriptive name.

Client Comment *The naming of token0 is derived from Uniswap, representing the token in the Uniswap pool that is sorted at index 0. isToken0 indicates whether EQU is the token sorted at index 0 in the Uniswap pool.*

```
32 /// @dev Indicates whether the EQU token is token0 or token1 in the
      ↪ Uniswap V3 Pool
   bool private immutable isToken0;
```

CVF-59. INFO

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

Recommendation The type of this argument should be more specific.

Client Comment *This address is only used for calculating the Pool address and, therefore, doesn't need to be specified as a more specific type.*

```
77 address _v3PoolFactory,
```

CVF-60. FIXED

- **Category** Suboptimal
- **Source** FeeDistributor.sol

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

```
96 uint16[] calldata _periods,
   uint16[] calldata _multipliers
```

CVF-61. INFO

- **Category** Procedural
- **Source** FeeDistributor.sol

Recommendation Consider using the ‘<interface>.<function>.selector” syntax instead of a hardcoded value.

Client Comment *These two functions have standard ERC20 function signatures and won’t change, so there’s no need to define an interface.*

```
240 // The value 0x40c10f19 represents the function selector for the '
       ↪ mint(address,uint256)' function.
    Address.functionCall(address(veEQU), abi.encodeWithSelector(0
       ↪ x40c10f19, _receiver, _amount));
```

```
285 // The value 0x9dc29fac represents the function selector for the '
       ↪ burn(address,uint256)' function.
    Address.functionCall(address(veEQU), abi.encodeWithSelector(0
       ↪ x9dc29fac, _owner, totalStakedAmount));
```

CVF-62. FIXED

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

Description Over-/underflow is possible here.

Recommendation Consider using checked math.

```
322 perShareGrowthX64 - stakeInfoCache.perShareGrowthX64,
    stakeInfoCache.amount * stakeInfoCache.multiplier,
```

CVF-63. FIXED

- **Category** Suboptimal
- **Source** FeeDistributor.sol

Recommendation It would be more efficient to use a shift instead of a division here.

```
341 amount = (_architectPerShareGrowthX64 - architectPerShareGrowthX64s[
       ↪ tokenID]) / Constants.Q64;
```

CVF-64. INFO

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

Description The argument names are confusing, as EQU and WETH could be either token 0 or token 1.

Recommendation Consider using different names, such as “baseToken” and “quoteToken”.

Client Comment *The naming of token0 and token1 is derived from Uniswap, and here we maintain consistency with that convention. However, it’s important to note that token0 and token1 do not necessarily represent the base token and quote token. Instead, they indicate the sorting order of the two tokens in the Uniswap pool.*

```
420 function _validateTokenPair(address _token0, address _token1)
        ↪ private view {
        if (address(EQU) != _token0 || address(WETH) != _token1) revert
            ↪ InvalidUniswapV3PositionNFT();
    }
```

CVF-65. INFO

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

Recommendation Event are usually named via nouns, such as “PluginApproval” or “Plugin-Revocation”.

Client Comment *Ignore.*

```
10 event PluginApproved(address indexed account, address indexed plugin
       ↪ );

15 event PluginRevoked(address indexed account, address indexed plugin)
       ↪ ;
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** IPluginManager.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
18 error PluginAlreadyRegistered();
20 error PluginNotRegistered();
22 error PluginAlreadyApproved();
24 error PluginNotApproved();
```

CVF-67. FIXED

- **Category** Procedural
- **Source** IPluginManager.sol

Recommendation This function should emit some event and this event should be declared in this interface.

```
29 function registerPlugin(address plugin) external;
```

CVF-68. INFO

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

Recommendation Events are usually named via nouns, such as “MinExecutionFee” or “OrderExecutorUpdate”.

```
9   event MinExecutionFeeUpdated(uint256 minExecutionFee);
14  event OrderExecutorUpdated(address account, bool active);
28  event IncreaseOrderCreated(
45  event IncreaseOrderUpdated(uint256 orderIndex, uint160
      ↪ triggerMarketPriceX96, uint160 acceptableTradePriceX96);
50  event IncreaseOrderCancelled(uint256 orderIndex, address payable
      ↪ feeReceiver);
56  event IncreaseOrderExecuted(uint256 orderIndex, uint160
      ↪ marketPriceX96, address payable feeReceiver);
73  event DecreaseOrderCreated(
91  event DecreaseOrderUpdated(uint256 orderIndex, uint160
      ↪ triggerMarketPriceX96, uint160 acceptableTradePriceX96);
96  event DecreaseOrderCancelled(uint256 orderIndex, address feeReceiver
      ↪ );
102 event DecreaseOrderExecuted(uint256 orderIndex, uint160
      ↪ marketPriceX96, address payable feeReceiver);
```

CVF-69. FIXED

- **Category** Suboptimal
- **Source** IOrderBook.sol

Recommendation The “account” parameter should be indexed.

```
14  event OrderExecutorUpdated(address account, bool active);
```

CVF-70. FIXED

- **Category** Suboptimal
- **Source** IOrderBook.sol

Recommendation These parameters should be indexed.

```
29 address account,
30 IPool pool,
```

```
38 uint256 orderIndex
```

```
74 address account,
   IPool pool,
```

```
84 uint256 orderIndex
```

CVF-71. FIXED

- **Category** Suboptimal
- **Source** IOrderBook.sol

Recommendation The “orderIndex” parameter should be indexed.

```
45 event IncreaseOrderUpdated(uint256 orderIndex, uint160
      ↪ triggerMarketPriceX96, uint160 acceptableTradePriceX96);
```

```
50 event IncreaseOrderCancelled(uint256 orderIndex, address payable
      ↪ feeReceiver);
```

```
56 event IncreaseOrderExecuted(uint256 orderIndex, uint160
      ↪ marketPriceX96, address payable feeReceiver);
```

```
91 event DecreaseOrderUpdated(uint256 orderIndex, uint160
      ↪ triggerMarketPriceX96, uint160 acceptableTradePriceX96);
```

```
96 event DecreaseOrderCancelled(uint256 orderIndex, address feeReceiver
      ↪ );
```

```
102 event DecreaseOrderExecuted(uint256 orderIndex, uint160
      ↪ marketPriceX96, address payable feeReceiver);
```

```
111 error OrderNotExists(uint256 orderIndex);
```

CVF-72. INFO

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

Description This function should emit some event and this event should be declared in this interface.

Client Comment *The parameter is not subject to frequent changes and is loosely related to the business logic. Additionally, this value can be obtained by reading the contract.*

```
159 function updateExecutionGasLimit(uint256 executionGasLimit) external
    ↪ ;
```

CVF-73. INFO

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

Recommendation Events are usually named via nouns, such as “MinExecutionFee” or “PositionExecutorUpdate”.

```
9   event MinExecutionFeeUpdated(uint256 minExecutionFee);

14  event PositionExecutorUpdated(address account, bool active);

20  event DelayValuesUpdated(uint32 minBlockDelayExecutor, uint32
        ↪ minTimeDelayPublic, uint32 maxTimeDelay);

29  event OpenLiquidityPositionCreated(

41  event OpenLiquidityPositionCancelled(uint128 index, address payable
        ↪ executionFeeReceiver);

46  event OpenLiquidityPositionExecuted(uint128 index, address payable
        ↪ executionFeeReceiver);

55  event CloseLiquidityPositionCreated(

67  event CloseLiquidityPositionCancelled(uint128 index, address payable
        ↪  executionFeeReceiver);

72  event CloseLiquidityPositionExecuted(uint128 index, address payable
        ↪ executionFeeReceiver);

82  event AdjustLiquidityPositionMarginCreated(

95  event AdjustLiquidityPositionMarginCancelled(uint128 index, address
        ↪ payable executionFeeReceiver);

100 event AdjustLiquidityPositionMarginExecuted(uint128 index, address
        ↪ payable executionFeeReceiver);

108 event IncreaseRiskBufferFundPositionCreated(
```

(119, 124, 133, 145, 150, 161, 175, 180, 192, 207, 212)

CVF-74. FIXED

- **Category** Suboptimal
- **Source** IPositionRouter.sol

Recommendation The “account” parameter should be indexed.

```
14 event PositionExecutorUpdated(address account, bool active);
```

CVF-75. FIXED

- **Category** Suboptimal
- **Source** IPositionRouter.sol

Recommendation These parameters should be indexed.

```
30  address account,
    IPool pool,
35  uint128 index
56  address account,
    IPool pool,
61  uint128 index
83  address account,
    IPool pool,
89  uint128 index
109 address account,
110 IPool pool,
113 uint128 index
134 address account,
    IPool pool,
139 uint128 index
162 address account,
    IPool pool,
169 uint128 index
193 address account,
    IPool pool,
201 uint128 index
```

CVF-76. FIXED

- **Category** Suboptimal
- **Source** IPositionRouter.sol

Recommendation The “index” parameter should be indexed.

```
41 event OpenLiquidityPositionCancelled(uint128 index, address payable
   ↪ executionFeeReceiver);

46 event OpenLiquidityPositionExecuted(uint128 index, address payable
   ↪ executionFeeReceiver);

67 event CloseLiquidityPositionCancelled(uint128 index, address payable
   ↪  executionFeeReceiver);

72 event CloseLiquidityPositionExecuted(uint128 index, address payable
   ↪ executionFeeReceiver);

95 event AdjustLiquidityPositionMarginCancelled(uint128 index, address
   ↪ payable executionFeeReceiver);

100 event AdjustLiquidityPositionMarginExecuted(uint128 index, address
   ↪ payable executionFeeReceiver);

119 event IncreaseRiskBufferFundPositionCancelled(uint128 index, address
   ↪  payable executionFeeReceiver);

124 event IncreaseRiskBufferFundPositionExecuted(uint128 index, address
   ↪ payable executionFeeReceiver);

145 event DecreaseRiskBufferFundPositionCancelled(uint128 index, address
   ↪  payable executionFeeReceiver);

150 event DecreaseRiskBufferFundPositionExecuted(uint128 index, address
   ↪ payable executionFeeReceiver);

175 event IncreasePositionCancelled(uint128 index, address payable
   ↪ executionFeeReceiver);

180 event IncreasePositionExecuted(uint128 index, address payable
   ↪ executionFeeReceiver);

207 event DecreasePositionCancelled(uint128 index, address payable
   ↪ executionFeeReceiver);
```

(212)

CVF-77. INFO

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

Description This function should emit some event and this event should be declared in this interface.

Client Comment *The parameter is not subject to frequent changes and is loosely related to the business logic. Additionally, this value can be obtained by reading the contract.*

```
322 function updateExecutionGasLimit(uint160 executionGasLimit) external
    ↪ ;
```

CVF-78. FIXED

- **Category** Documentation
- **Source** IPositionRouter.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or explaining in the documentation comment.

```
347 function executeOpenLiquidityPosition(uint128 index, address payable
    ↪  executionFeeReceiver) external returns (bool);
```

CVF-79. INFO

- **Category** Procedural
- **Source** PluginManager.sol

Description We didn't review this file.

Client Comment *This contract is very simple and doesn't require an audit.*

```
4 import "../governance/Governable.sol";
```

CVF-80. INFO

- **Category** Suboptimal
- **Source** PluginManager.sol

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

Client Comment *This can make the code clearer.*

```
8 mapping(address => bool) public override registeredPlugins;
  mapping(address => mapping(address => bool)) private pluginApprovals
    ↪ ;
```

CVF-81. INFO

- **Category** Procedural
- **Source** Router.sol

Description We didn't review this file.

Client Comment *This contract is very simple and doesn't require an audit.*

```
6 import "../tokens/interfaces/IEFC.sol";
```

CVF-82. INFO

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

Recommendation The default value for this variable should be a named constant.

Client Comment *Using a magic number here can enhance readability. Additionally, this value is only used in one place and is initialized with this value, but it may be updated later on.*

```
16 uint256 public executionGasLimit = 1_000_000;
```

CVF-83. INFO

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

Description These events are emitted even if nothing actually changed.

Client Comment *Generally, the same value wii not be updated, and adding additional checks would make the code bloated.*

```
39 emit MinExecutionFeeUpdated(_minExecutionFee);
```

```
45 emit OrderExecutorUpdated(_account, _active);
```

CVF-84. INFO

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

Description This function should emit some event.

Client Comment *The parameter is not subject to frequent changes and is loosely related to the business logic. Additionally, this value can be obtained by reading the contract.*

```
49 function updateExecutionGasLimit(uint256 _executionGasLimit)
   ↪ external override onlyGov {
```

CVF-85. INFO

- **Category** Procedural
- **Source** PositionRouter.sol

Description We didn't review these files.

Client Comment *This file is a copy of the "@openzeppelin" contracts, so it doesn't require an audit.*

```
5 import "../libraries/SafeCast.sol";
import "../libraries/ReentrancyGuard.sol";
```

CVF-86. INFO

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

Recommendation The default values should be named constants.

Client Comment *Using a magic number here can enhance readability. Additionally, this value is only used in one place and is initialized with this value, but it may be updated later on.*

```
24 uint32 public minTimeDelayPublic = 180;
   uint32 public maxTimeDelay = 1800;
   uint160 public executionGasLimit = 1_000_000;
```

CVF-87. INFO

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

Description These events are emitted even if nothing actually changed.

Client Comment *Generally, the same value wii not be updated, and adding additional checks would make the code bloated.*

```
74 emit PositionExecutorUpdated(_account, _active);
```

```
85 emit DelayValuesUpdated(_minBlockDelayExecutor, _minTimeDelayPublic,
   ↪  _maxTimeDelay);
```

```
90 emit MinExecutionFeeUpdated(_minExecutionFee);
```

CVF-88. INFO

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

Description This function should emit some event.

Client Comment *The parameter is not subject to frequent changes and is loosely related to the business logic. Additionally, this value can be obtained by reading the contract.*

```
93 function updateExecutionGasLimit(uint160 _executionGasLimit)
   ↪ external override onlyGov {
```

CVF-89. INFO

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

Description This function always return true.

Recommendation Consider returning nothing.

Client Comment *Not really, when ‘minBlockDelayExecutor‘ is not zero, false might be returned.*

```
833 ) internal view returns (bool) {
```

CVF-90. FIXED

- **Category** Suboptimal
- **Source** RewardCollector.sol

Recommendation It would be more efficient to cache the array lengths in local variables.

```
37 for (uint256 i; i < _pools.length; ++i) {
```

```
39     for (uint256 j; j < _referralTokens.length; ++j) {
```

CVF-91. INFO

- **Category** Procedural
- **Source** IPriceFeed.sol

Description We didn’t review this file.

Client Comment *This file is the interface provided by Chainlink and does not require an audit.*

```
4 import "./IChainLinkAggregator.sol";
```

CVF-92. INFO

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

Recommendation Events are usually named via nouns, such as “Price”.

Client Comment *Ignore.*

```
13 event PriceUpdated(IERC20 indexed token, uint160 priceX96, uint160
   ↪ minPriceX96, uint160 maxPriceX96);
```

```
21 event MaxCumulativeDeltaDiffExceeded(
```

CVF-93. FIXED

- **Category** Suboptimal
- **Source** IPriceFeed.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
41 error InvalidReferencePrice();
```

```
53 error GracePeriodNotOver();
```

CVF-94. INFO

- **Category** Procedural
- **Source** PriceFeed.sol

Description We didn’t review these files.

Client Comment *This file is a copy of the “@openzeppelin” contracts, so it doesn’t require an audit.*

```
4 import "../libraries/SafeCast.sol";
```

```
8 import "../governance/Governable.sol";
```

CVF-95. FIXED

- **Category** Readability
- **Source** PriceFeed.sol

Recommendation This value could be rendered as “30 minutes” in Solidity.

```
18 uint256 public constant GRACE_PERIOD_TIME = 1800; // 30 min
```

CVF-96. FIXED

- **Category** Readability
- **Source** PriceFeed.sol

Recommendation These values could be rendered as: 100e3, 1 minutes, 1 minutes

```
40 (slot.maxDeviationRatio, slot.cumulativeRoundDuration, slot.
     ↪ updateTxTimeout) = (100 * 1000, 60, 60);
```

CVF-97. FIXED

- **Category** Suboptimal
- **Source** PriceFeed.sol

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

```
45 IERC20[] calldata _tokens,
   uint160[] calldata _priceX96s
```

```
84 IERC20[] calldata _tokens,
   uint160[] calldata _priceX96s,
```

CVF-98. INFO

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

Description These functions should emit some events.

Client Comment *The parameter is not subject to frequent changes and is loosely related to the business logic. Additionally, this value can be obtained by reading the contract.*

```
146 function setUpdater(address _account, bool _active) external
    ↪ override onlyGov {

156 function setRefPriceFeeds(IERC20 _token, IChainLinkAggregator
    ↪ _priceFeed) external override onlyGov {

161 function setSequencerUptimeFeed(IChainLinkAggregator
    ↪ _sequencerUptimeFeed) external override onlyGov {

166 function setRefHeartbeatDuration(IERC20 _token, uint32 _duration)
    ↪ external override onlyGov {

171 function setMaxDeviationRatio(uint32 _maxDeviationRatio) external
    ↪ override onlyGov {

176 function setCumulativeRoundDuration(uint32 _cumulativeRoundDuration)
    ↪  external override onlyGov {

181 function setMaxCumulativeDeltaDiffs(IERC20 _token, uint64
    ↪ _maxCumulativeDeltaDiff) external override onlyGov {

186 function setRefPriceExtraSample(uint32 _refPriceExtraSample)
    ↪ external override onlyGov {

191 function setUpdateTxTimeout(uint32 _updateTxTimeout) external
    ↪ override onlyGov {
```

CVF-99. FIXED

- **Category** Suboptimal
- **Source** PriceFeed.sol

Recommendation As both, numerator and denominator here are powers of 10, the "mulDiv" call could be replaced here with plain division and multiplication: uint256(10)**USD_DECIMALS >= _mgnification ? _price * (uint256(10)**USD_DECIMALS / _magnification) : _price / (_magnification / uint256(10)**USD_DECIMALS)

```
265 unchecked { _price = Math.mulDiv(_price, uint256(10) ** USD_DECIMALS
    ↪ , _magnification); }
```

CVF-100. INFO

- **Category** Procedural
- **Source** Math.sol

Description The library name is inconsistent with the file name.

Recommendation Consider making them consistent.

Client Comment *To avoid conflicts with @openzeppelin's Math.*

```
10 library M {
```

CVF-101. INFO

- **Category** Suboptimal
- **Source** Math.sol

Recommendation This line is redundant, as b==0 would anyway cause division by zero later.

Client Comment *Removing this check would result in incorrect behavior when a == 0.*

```
22 if (b == 0) return a / b;
```

CVF-102. INFO

- **Category** Suboptimal
- **Source** Math.sol

Description The value “mulmod(x, y, denominator)” was already calculated inside the “mulDiv” function.

Recommendation Consider refactoring to reuse this already calculated value.

Client Comment *The value “mulmod(x, y, denominator)” is not calculated in some cases.*

```
48 if (mulmod(x, y, denominator) > 0) resultUp += 1;
```

CVF-103. INFO

- **Category** Procedural
- **Source** PositionUtil.sol

Description We didn’t review this file.

Client Comment *This file is a copy of the “@openzeppelin” contracts, so it doesn’t require an audit.*

```
6 import "./SafeCast.sol";
```

CVF-104. FIXED

- **Category** Suboptimal
- **Source** LiquidityPositionUtil.sol

Description The expression “-liquidityDelta” is calculated twice.

Recommendation Consider calculating once and reusing.

```
67 _metrics.liquidity -= uint128(uint256(-_liquidityDelta));
```

```
70     uint256(-_liquidityDelta);
```

CVF-105. INFO

- **Category** Suboptimal
- **Source** LiquidityPositionUtil.sol

Description While the “mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, when the denominator is a power of 2, the division could be replaced with a shift.

Recommendation Consider implementing such an optimized function.

Client Comment *The result of “x * y” may exceed “type(uint256).max”.*

```
85 realizedProfit = Math.mulDiv(deltaX64, _positionCache.liquidity,
      ↪ Constants.Q64);
```

CVF-106. INFO

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

Recommendation Events are usually named via nouns, such as “PremiumRate” or “LiquidityBufferSize”.

Client Comment *Ignore.*

```
28 event PremiumRateChanged(uint128 premiumRateAfterX96);
```

```
31 event LiquidationBufferNetSizeChanged(uint8 index, uint128
      ↪ netSizeAfter);
```

CVF-107. INFO

- **Category** Suboptimal
- **Source** PriceUtil.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *‘MaxPremiumRate‘ is a fixed configuration item and can be obtained through various other means. Adding a fixed value doesn’t have much significance.*

```
34 error MaxPremiumRateExceeded();
```

```
37 error InvalidSizeDelta();
```

CVF-108. FIXED

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

Recommendation A better name for this error would be “ZeroSizeDelta”.

```
37 error InvalidSizeDelta();
```

CVF-109. FIXED

- **Category** Procedural
- **Source** PriceUtil.sol

Recommendation Brackets are redundant.

```
82 globalPositionCache.netSize += (_sizeDelta - totalBufferUsed);
```

```
95 globalPositionCache.netSize -= (_sizeDelta - sizeLeft -
   ↪ totalBufferUsed);
```

CVF-110. FIXED

- **Category** Suboptimal
- **Source** PriceUtil.sol

Description The value “_step.current.size - _step.to.size” is calculated twice.

Recommendation Consider calculating once and reusing.

```
279 reached = _step.sizeLeft >= _step.current.size - _step.to.size;
280 sizeUsed = reached ? _step.current.size - _step.to.size : _step.
   ↪ sizeLeft;
```

CVF-111. FIXED

- **Category** Suboptimal
- **Source** PriceUtil.sol

Description The value "_step.to.size - _step.current.size" is calculated twice.

Recommendation Consider calculating once and reusing.

```
282 reached = _step.sizeLeft >= _step.to.size - _step.current.size;
    sizeUsed = reached ? _step.to.size - _step.current.size : _step.
       ↪ sizeLeft;
```

CVF-112. INFO

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

Recommendation Events are usually named via nouns, such as "GlobalFundingRateSample".

Client Comment *Ignore.*

```
13 event GlobalFundingRateSampleAdjusted(uint16 sampleCountAfter,
       ↪ int176 cumulativePremiumRateAfterX96);
```

```
16 event GlobalRiskBufferFundChanged(int256 riskBufferFundAfter);
```

CVF-113. INFO

- **Category** Documentation
- **Source** FundingRateUtil.sol

Description The code uses a named constant, while the comment contains a particular value. If constant. value will ever be changed, the comment will become incorrect.

Recommendation Consider removing the value from the comment.

Client Comment *Adding comments can make this part of the logic clearer.*

```
35 uint64 maxSamplingTime = lastAdjustFundingRateTime + Constants.
       ↪ ADJUST_FUNDING_RATE_INTERVAL;
```

```
37 // At most 1 hour of premium rate sampling
```

CVF-114. INFO

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

Recommendation Events are usually named via nouns, such as “LiquidityRewardDebt” or “LiquidityRewardCollection”.

Client Comment *Ignore*.

```
13  event LiquidityRewardDebtChanged(IPool indexed pool, address indexed
        ↪  account, uint256 rewardDebtDelta);

20  event LiquidityRewardCollected(

31  event RiskBufferFundRewardDebtChanged(IPool indexed pool, address
        ↪ indexed account, uint256 rewardDebtDelta);

38  event RiskBufferFundRewardCollected(

49  event PoolLiquidityRewardGrowthIncreased(IPool indexed pool, uint256
        ↪  rewardDelta, uint128 rewardGrowthAfterX64);

58  event PoolReferralTokenRewardGrowthIncreased(

73  event PoolReferralParentTokenRewardGrowthIncreased(

85  event PoolRiskBufferFundRewardGrowthIncreased(

94  event PoolRewardUpdated(IPool indexed pool, uint160 rewardPerSecond)
        ↪ ;

100 event ReferralLiquidityRewardDebtChanged(

110 event ReferralPositionRewardDebtChanged(IPool indexed pool, uint256
        ↪ indexed referralToken, uint256 rewardDebtDelta);

117 event ReferralRewardCollected(IPool[] pools, uint256[]
        ↪ referralTokens, address receiver, uint256 rewardDebt);

121 event ConfigChanged(Config newConfig);

125 event RewardCapChanged(uint128 rewardCapAfter);
```

CVF-115. INFO

- **Category** Suboptimal
- **Source** IRewardFarm.sol

Description Logging several pools as an array makes it impossible to index by them.

Recommendation Consider emitting a separate event for each pool.

Client Comment *Combining the sends will save gas, make the logic clearer, and won't affect event indexing in the subgraph.*

```
21     IPool[] pools,
```

```
39     IPool[] pools,
```

```
117 event ReferralRewardCollected(IPool[] pools, uint256[]
    ↪ referralTokens, address receiver, uint256 rewardDebt);
```

CVF-116. FIXED

- **Category** Suboptimal
- **Source** IRewardFarm.sol

Recommendation The "receiver" parameter should be indexed.

```
117 event ReferralRewardCollected(IPool[] pools, uint256[]
    ↪ referralTokens, address receiver, uint256 rewardDebt);
```

CVF-117. FIXED

- **Category** Suboptimal
- **Source** IRewardFarm.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
134 error InvalidMintTime();
```

```
136 error InvalidMiningRate();
```

CVF-118. INFO

- **Category** Procedural
- **Source** RewardFarm.sol

Description We didn't review this library.

Client Comment *This file is a copy of the "@openzeppelin" contracts, so it doesn't require an audit.*

```
4 import "../libraries/SafeCast.sol";
```

CVF-119. FIXED

- **Category** Documentation
- **Source** RewardFarm.sol

Description The number. format for this variable is unclear.

Recommendation Consider documenting.

```
31 uint32 public immutable referralMultiplier;
```

CVF-120. INFO

- **Category** Suboptimal
- **Source** RewardFarm.sol

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

Client Comment *Each mapping represents a separate business, keeping them separate makes it clear. Any modifications to each business only update that business's data.*

```
45 mapping(address => RiskBufferFundReward) public
      ↪ riskBufferFundRewards;
   mapping(address => bool) public alreadyBoundReferralTokens;
   mapping(address => LiquidityReward) public liquidityRewards;
   mapping(address => Position) public positions;
```

CVF-121. FIXED

- **Category** Documentation
- **Source** RewardFarm.sol

Description The semantics of the keys for these mappings is unclear.

Recommendation Consider documenting.

```
45 mapping(address => RiskBufferFundReward) public
      ↪ riskBufferFundRewards;
   mapping(address => bool) public alreadyBoundReferralTokens;
   mapping(address => LiquidityReward) public liquidityRewards;
   mapping(address => Position) public positions;
```

CVF-122. FIXED

- **Category** Suboptimal
- **Source** RewardFarm.sol

Recommendation This code is an overkill. Just convert everything to 256 bits, do addition ahead of subtraction and then convert back to 128 bits.

```
153 poolReward.riskBufferFundLiquidity = _liquidityAfter >
      ↪ liquidityBefore
       ? poolReward.riskBufferFundLiquidity + (_liquidityAfter -
          ↪ liquidityBefore).toUint128()
       : poolReward.riskBufferFundLiquidity - (liquidityBefore -
          ↪ _liquidityAfter).toUint128();
```

CVF-123. INFO

- **Category** Suboptimal
- **Source** RewardFarm.sol

Description While the “mulDiv” function is very efficient in generic case, for specific cases better approaches do exist. For example, then the numerator or the denominator is a power of 2, shift could be used instead of multiplication or division. When the denominator is a compile-time constant, its modular reciprocal could be precomputed.

Recommendation Consider implementing such optimized functions.

Client Comment *The result of “x * y” may exceed “type(uint256).max”.*

```
174 uint128 positionAfter = Math.mulDiv(_sizeAfter, _entryPriceAfterX96,
      ↪  Constants.Q96).toUint128();

428 if (_totalLiquidity != 0) perShareGrowthX64 = Math.mulDiv(_amount,
      ↪ Constants.Q64, _totalLiquidity).toUint128();

638     ? Math.mulDiv(_reward.liquidity, referralMultiplier, Constants.
          ↪ BASIS_POINTS_DIVISOR)

694 amount = Math.mulDiv(_reward, _rate, Constants.BASIS_POINTS_DIVISOR)
      ↪ ;

716 unchecked { rewardDebt = Math.mulDiv(_globalRewardGrowthX64 -
      ↪ _rewardGrowthX64, _liquidity, Constants.Q64); }
```

CVF-124. INFO

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

Recommendation Event are usually named via nouns, such as “USDToken” or “USDTokenConfig”.

Client Comment *Ignore.*

```
11 event USDEnabled(IERC20 indexed usd);
```

CVF-125. FIXED

- **Category** Suboptimal
- **Source** IConfigurable.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
28 error TokenAlreadyEnabled();
30 error InvalidMaxRiskRatePerLiquidityPosition();
32 error InvalidMaxLeveragePerLiquidityPosition();
34 error InvalidMaxLeveragePerPosition();
36 error InvalidLiquidationFeeRatePerPosition();
38 error InvalidInterestRate();
40 error InvalidMaxFundingRate();
42 error InvalidTradingFeeRate();
44 error InvalidLiquidityFeeRate();
46 error InvalidProtocolFeeRate();
48 error InvalidReferralReturnFeeRate();
50 error InvalidReferralParentReturnFeeRate();
52 error InvalidReferralDiscountRate();
54 error InvalidFeeRate();
56 error InvalidMaxPriceImpactLiquidity();
59 error InvalidVerticesLength();
62 error InvalidLiquidationVertexIndex();
```

CVF-126. INFO

- **Category** Documentation
- **Source** IConfigurable.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

Client Comment *Comments already exist on the corresponding functions.*

```
69 uint64 minMarginPerLiquidityPosition;
70 uint32 maxRiskRatePerLiquidityPosition;
   uint32 maxLeveragePerLiquidityPosition;
```

```
73 uint64 minMarginPerPosition;
   uint32 maxLeveragePerPosition;
   uint32 liquidationFeeRatePerPosition;
```

```
77 uint64 liquidationExecutionFee;
   uint32 interestRate;
   uint32 maxFundingRate;
```

```
83 uint32 tradingFeeRate;
   uint32 liquidityFeeRate;
   uint32 protocolFeeRate;
   uint32 referralReturnFeeRate;
   uint32 referralParentReturnFeeRate;
   uint32 referralDiscountRate;
```

```
92 uint32 balanceRate;
   uint32 premiumRate;
```

```
97 uint128 maxPriceImpactLiquidity;
```

CVF-127. INFO

- **Category** Suboptimal
- **Source** IPoolErrors.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *The name can already clearly express the error type.*

```
8  error InvalidLiquidityToOpen();
```

```
14 error InsufficientMargin();
```

```
28 error InsufficientGlobalLiquidity();
```

CVF-128. INFO

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

Recommendation Events are usually named via nouns, such as “FundingRateGrowth” or “PositionIncrease”.

Client Comment *Ignore.*

```
15 event FundingRateGrowthAdjusted(
```

```
34 event PositionIncreased(
```

```
59 event PositionDecreased(
```

```
85 event PositionLiquidated(
```

CVF-129. INFO

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

Recommendation Events are usually named via nouns, such as “GlobalUnrealizedLoss-Metrics” or “LiquidityPosition”.

Client Comment *Ignore*.

```
16  event GlobalUnrealizedLossMetricsChanged(
30  event LiquidityPositionOpened(
46  event LiquidityPositionClosed(
61  event LiquidityPositionMarginAdjusted(
77  event LiquidityPositionLiquidated(
91  event GlobalLiquidityPositionNetPositionAdjusted(
100 event GlobalLiquidityPositionRealizedProfitGrowthChanged(uint256
      ↪ realizedProfitGrowthAfterX64);
105 event GlobalRiskBufferFundGovUsed(address indexed receiver, uint128
      ↪ riskBufferFundDelta);
108 event GlobalRiskBufferFundChanged(int256 riskBufferFundAfter);
114 event RiskBufferFundPositionIncreased(address indexed account,
      ↪ uint128 liquidityAfter, uint64 unlockTimeAfter);
120 event RiskBufferFundPositionDecreased(address indexed account,
      ↪ uint128 liquidityAfter, address receiver);
```

CVF-130. INFO

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

Recommendation Events are usually named via nouns, such as “PriceVertex” or “ProtocolFeeIncrease”.

Client Comment *Ignore.*

```
28 event PriceVertexChanged(uint8 index, uint128 sizeAfter, uint128
   ↪ premiumRateAfterX96);

32 event ProtocolFeeIncreased(uint128 amount);

36 event ProtocolFeeCollected(uint128 amount);

44 event ReferralFeeIncreased(

56 event ReferralFeeCollected(uint256 indexed referralToken, address
   ↪ indexed receiver, uint256 amount);
```

CVF-131. INFO

- **Category** Suboptimal
- **Source** IPool.sol

Recommendation The value “7” should be a named constant.

Client Comment *The current compiler version doesn’t support “PriceVertex[Constants.VERTEX_NUM]”, so hardcoding is the only option.*

```
82 PriceVertex[7] memory priceVertices,

86 uint128[7] memory liquidationBufferNetSizes
```

CVF-132. INFO

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

Recommendation Events are usually named via nouns, such as “Pool”.

Client Comment *Ignore.*

```
17 event PoolCreated(IPool indexed pool, IERC20 token, IERC20 usd);
```

CVF-133. FIXED

- **Category** Suboptimal
- **Source** IPoolFactory.sol

Recommendation All parameters should be indexed.

```
17 event PoolCreated(IPool indexed pool, IERC20 token, IERC20 usd);
```

CVF-134. FIXED

- **Category** Suboptimal
- **Source** IPoolFactory.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
23 error PoolAlreadyExists();
```

CVF-135. INFO

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

Description Despite the name, this function returns a single pool.

Recommendation Consider renaming.

Client Comment *This function is automatically generated by the Solidity compiler.*

```
54 function pools(IERC20 token) external view returns (IPool pool);
```

CVF-136. INFO

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

Recommendation The argument type should be “IPool”.

Client Comment *The purpose of this function is to check if an address is an “IPool”, so it would seem strange if the parameter type is “IPool”.*

```
59 function isPool(address pool) external view returns (bool);
```

CVF-137. INFO

- **Category** Procedural
- **Source** Configurable.sol

Description We didn’t review these files.

Client Comment *This file is a copy of the “@openzeppelin” contracts, so it doesn’t require an audit.*

```
5 import "../libraries/ReentrancyGuard.sol";
import "../governance/Governable.sol";
```

CVF-138. INFO

- **Category** Suboptimal
- **Source** Configurable.sol

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

Client Comment *This can make the code clearer.*

```
12 mapping(IERC20 => TokenConfig) public override tokenConfigs;
```

```
14 mapping(IERC20 => TokenFeeRateConfig) public override
   ↪ tokenFeeRateConfigs;
mapping(IERC20 => TokenPriceConfig) private tokenPriceConfigs0;
```

CVF-139. INFO

- **Category** Suboptimal
- **Source** Configurable.sol

Description Using a business-level property as a low-level indicator of entity existence is a bad practice. Business constraints may change in the future, and entities with zero value for this property could become eligible.

Recommendation Consider using a separate dedication low-level flag.

Client Comment *Storing fewer values can save gas.*

```
41 if (tokenConfigs[_token].maxLeveragePerLiquidityPosition > 0) revert
      ↪  TokenAlreadyEnabled();
```

```
55 if (tokenConfigs[_token].maxLeveragePerLiquidityPosition == 0)
      ↪ revert TokenNotEnabled();
```

```
61 return tokenConfigs[_token].maxLeveragePerLiquidityPosition != 0;
```

CVF-140. INFO

- **Category** Suboptimal
- **Source** Pool.sol

Recommendation This line could be simplified using the “+=” operator.

Client Comment *Using values in memory can save gas.*

```
114 globalLiquidityPosition.liquidity = globalPositionCache.liquidity +
      ↪ _liquidity;
```

CVF-141. INFO

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

Description The reasoning is unclear.

Recommendation Consider illustrating with an example.

Client Comment *You need the whitepaper as prerequisite knowledge here.*

```
761 // If the vertex represented by end is the same as the vertex
      ↪ represented by end + 1,
    // then the vertices in the range (start, LATEST_VERTEX] need to be
      ↪ updated
```

CVF-142. INFO

- **Category** Procedural
- **Source** Pool.sol

Recommendation Brackets around multiplication are redundant.

Client Comment *Adding brackets can make the code clearer.*

```
952 uint256 balanceRateX96 = (Constants.Q96 * _balanceRate) / Constants.
    ↪ BASIS_POINTS_DIVISOR;
```

```
955 premiumRateX96 = uint128((Constants.Q96 * _premiumRate) / Constants.
    ↪ BASIS_POINTS_DIVISOR);
```

CVF-143. INFO

- **Category** Suboptimal
- **Source** Pool.sol

Recommendation These calculations could be replaced with multiplications by a precomputed value 2^96 / 1e8, that is still big enough to maintain high precision when represented as an integer.

Client Comment *This adjustment will result in a loss of some precision.*

```
952 uint256 balanceRateX96 = (Constants.Q96 * _balanceRate) / Constants.
    ↪ BASIS_POINTS_DIVISOR;
```

```
955 premiumRateX96 = uint128((Constants.Q96 * _premiumRate) / Constants.
    ↪ BASIS_POINTS_DIVISOR);
```

CVF-144. INFO

- **Category** Procedural
- **Source** Pool.sol

Recommendation Brackets are redundant.

Client Comment *Adding brackets can make the code clearer.*

```
1005 unchecked { return _timestamp - (_timestamp % Constants.
     ↪ ADJUST_FUNDING_RATE_INTERVAL); }
```

CVF-145. INFO

- **Category** Procedural
- **Source** PoolFactory.sol

Description We didn't review this file.

Client Comment *This contract is very simple and doesn't require an audit.*

```
6 import "../governance/Governable.sol";
```

CVF-146. INFO

- **Category** Suboptimal
- **Source** PoolFactory.sol

Description Writing creation code into the storage is very expensive.

Recommendation Consider deploying it as a separate contract and reading from there using the "EXTCODECOPY" instruction.

Client Comment *Gas will only be consumed during the initial deployment.*

```
11 bytes public creationCode;
```

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