Report

v. 1.0

Customer

Gridex

Smart Contract Audit

Gridex Protocol

13th February 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	10.02.23	A. Zveryanskaya	Initial Draft
0.2	13.02.23	A. Zveryanskaya	Minor revision
1.0	13.02.23	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.

Gridex is the first fully on-chain order book trading protocol for Ethereum ecosystem.

3 Project scope

We were asked to review:

- Original Code #1
- Original Code #2
- Code with Fixes

Files:

core/		
Grid.sol	GridDeployer.sol	GridFactory.sol
PriceOracle.sol	TradingConfig.sol	
core/interfaces/		
ITradingConfig.sol	IWETHMinimum.sol	
core/libraries/		
BitMath.sol	BoundaryBitmap.sol	BoundaryMath.sol
BundleMath.sol	CallbackValidator.sol	FeeMath.sol
FixedPointX96.sol	FixedPointX128.sol	FixedPointX192.sol
GridAddress.sol	SwapMath.sol	Uint128Math.sol
Uint160Math.sol		
governance/		
GDX.sol	GridexGovernor.sol	TreasuryVester.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 21 major, and a few less important issues. All identified Major issues have been fixed or otherwise addressed in collaboration with the client.

	Active	Fixed
Major	10	11
Moderate	7	6
Minor	99	21

Fixed 38 out of 154 issues

6 Major Issues

CVF-1. FIXED

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Description The values calculated here may remain unused in certain scenarios.

Recommendation Consider calculating these values only when they are really needed.

```
64 uint160 boundaryLowerPriceX96 = BoundaryMath.getPriceX96AtBoundary(
      ↪ boundaryLower);
   uint160 boundaryUpperPriceX96 = BoundaryMath.getPriceX96AtBoundary(
      ↪ boundaryUpper);
```

CVF-2. INFO

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

Description This function divides its argument by “resolution” and then multiples the returned value by “resolution”.

Recommendation Consider removing the “resolution” argument and operating on compressed boundaries.

Client Comment *The arguament ”resolution” is needed in this function.*

```
if (boundary < 0  boundary % resolution != 0) {
// round towards negative infinity
--compressed;
}
```

```
96 int24 compressed = boundary / resolution;
```

```
111         ? (compressed - 1 - int24(uint24(bitPos - BitMath.
               ↪ mostSignificantBit(masked)))) * resolution
            : (compressed - 1 - int24(uint24(bitPos))) * resolution;
```

```
131         ? (compressed + 1 + int24(uint24(BitMath.leastSignificantBit
               ↪ (masked) - bitPos))) * resolution
            : (compressed + 1 + int24(uint24(type(uint8).max - bitPos)))
               ↪  * resolution;
```

CVF-3. INFO

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

Description Multiplication after division could lead to precision degradation.

Recommendation Consider performing division after multiplication.

Client Comment *Specfically divide first, then multiply to ensure that the amountOut is smaller.*

```
132 uint256 numerator = 2 * takerAmountInWithoutFee * (FixedPointX192.Q
    ↪ / priceCurrentX96);
```

CVF-4. INFO

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

Description There is no range check for the “priceX96” argument.

Recommendation Consider adding appropriate check.

Client Comment *All callers are already range checked, so there is no need for a double check.*

```
79 function getBoundaryAtPriceX96(uint160 priceX96) internal pure
    ↪ returns (int24 boundary) {
```

CVF-5. FIXED

- **Category** Procedural
- **Source** BitMath.sol

Recommendation Specifying an unbounded range of compiler versions is a bad practice as one cannot guarantee compatibility with future major releases.

```
2 pragma solidity >=0.5.0;
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** GridFactory.sol

Description Writing the same grid address into two storage slots looks like waste of gas.

Recommendation Consider writing only to the [token0][token1] slot handling misordered tokens in the “grids” getter.

Client Comment *Its easier for third party integration. The extra gas cost only exists when creating the grid.*

```
46 grids[tokenA][tokenB][resolution] = grid;
   grids[tokenB][tokenA][resolution] = grid;
```

CVF-7. FIXED

- **Category** Procedural
- **Source** PriceOracle.sol

Description In some cases, in order to check whether certain grid is registered, the “capacity” field is used, while in other cases, the “nextCapacity” field is used instead.

Recommendation Consider using consistent approach across the code.

```
29 // PO_AR: already registered
30 require(gridOracleStates[grid].capacity == 0, "PO_AR");
```

```
48 // PO_UR: unregistered grid
   require(stateCache.capacityNext >= 1, "PO_UR");
```

```
74 uint16 capacityOld = state.capacityNext;
   // PO_UR: unregistered grid
   require(capacityOld >= 1, "PO_UR");
```

```
98 // PO_UR: unregistered grid
   require(state.capacity >= 1, "PO_UR");
```

```
112 // PO_UR: unregistered grid
    require(state.capacity >= 1, "PO_UR");
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** PriceOracle.sol

Description This expression is calculated at every loop iteration.

Recommendation Consider calculating once before the loop.

```
121 gridPriceData[grid],
```

CVF-9. FIXED

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

Description Multiplication after division could lead to precision degradation.

Recommendation Consider performing division at the end of a calculation.

```
172 return beforePriceData.boundaryCumulative + afterBoundary * int56(
       ↪ uint56(delta));
```

CVF-10. INFO

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

Description There are no range checked for the parameters obtained here.

Recommendation Consider adding appropriate checks, such as that tokens are different, resolution is positive etc.

Client Comment *Paramater checks are guaranteed by GridFactory.*

```
67 (token0, token1, resolution, tradingConfig, priceOracle, weth9) =
       ↪ IGridDeployer(_msgSender()).parameters();
```

CVF-11. INFO

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

Client Comment *Overflow is allowed here. The oracle will still function correctly while overflow is occuring.*

```
287 slot0Cache.blockTimestamp != (blockTimestamp = uint32(block.
    ↪ timestamp))
```

CVF-12. FIXED

- **Category** Suboptimal
- **Source** Grid.sol

Description The “mulDiv” function looks like an overkill here, as all the arguments fit into 128 bits.

Recommendation Consider using plain multiplication and division instead.

Client Comment *New business logic: no longer taking in protocol fees.*

```
299 uint128 feeChannel = Math.mulDiv(state.feeProtocol, 8, 10).toUint128
    ↪ ();
```

CVF-13. INFO

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

Description Underflow here is prevented implicitly by a complex business logic. Such implicit stuff is error-prone as business-logic may change.

Recommendation Consider using checked math.

Client Comment *For the purpose of optimizing gas, no modification is needed. Since the contract is non-upgradeable, the business logic will not change once deployed.*

```
429 makerAmountRemaining = boundary.makerAmountRemaining - step.
    ↪ amountOut;
```

CVF-14. FIXED

- **Category** Suboptimal
- **Source** BundleMath.sol

Description In quite probable case when “amountOutUsed” equals “amountOut”, calling the “muldiv” function is an overkill.

Recommendation Consider handling this case separately.

```
35 parameters.amountInUsed = Math.mulDiv(parameters.amountOutUsed,
      ↪ amountIn, amountOut);
```

```
44     .mulDiv(parameters.amountOutUsed, takerFeeForMakerAmount,
          ↪ amountOut)
```

CVF-15. FIXED

- **Category** Suboptimal
- **Source** BundleMath.sol

Description The “mulDiv” function is not needed here, as all the arguments are 128-bit numbers.

Recommendation Consider using normal 256-bit multiplication and division instead.

```
35 parameters.amountInUsed = Math.mulDiv(parameters.amountOutUsed,
      ↪ amountIn, amountOut);
```

```
44     .mulDiv(parameters.amountOutUsed, takerFeeForMakerAmount,
          ↪ amountOut)
```

```
99 makerAmountOut = Math.mulDiv(makerAmountRaw, makerAmountRemaining,
      ↪ makerAmountTotal).toUint128();
```

```
101 takerAmountOut = Math.mulDiv(makerAmountRaw, takerAmountRemaining,
       ↪ makerAmountTotal).toUint128();
takerFeeAmountOut = Math.mulDiv(makerAmountRaw,
       ↪ takerFeeAmountRemaining, makerAmountTotal).toUint128();
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** BundleMath.sol

Description The “mulDiv” function spends lots of gas to calculate the reciprocal of the denominator. When the same denominator is used several times, the reciprocal could be calculated once and reusing.

Recommendation Consider refactoring to reuse the reciprocal.

```
99 makerAmountOut = Math.mulDiv(makerAmountRaw, makerAmountRemaining,
      ↪ makerAmountTotal).toUint128();
```

```
101 takerAmountOut = Math.mulDiv(makerAmountRaw, takerAmountRemaining,
      ↪ makerAmountTotal).toUint128();
    takerFeeAmountOut = Math.mulDiv(makerAmountRaw,
      ↪ takerFeeAmountRemaining, makerAmountTotal).toUint128();
```

CVF-17. INFO

- **Category** Flaw
- **Source** FeeMath.sol

Description There is no check to ensure that makerFeePips < 0.

Recommendation Consider adding such a check.

Client Comment *Paramater checks are guaranteed by GridFactory.*

```
21 uint24 makerFeePipsAbs = uint24(-makerFeePips);
```

CVF-18. INFO

- **Category** Procedural
- **Source** IGridEvents.sol

Description Different types are used for token amounts.

Recommendation Consider using the same type for consistency.

Client Comment *This is adapated to different functions and cases. Therefore, data types are inconsistent (saves more gas).*

```
26      uint128 amount
```

```
36      uint128 makerAmountOut,
        uint128 takerAmountOut,
        uint128 takerFeeAmountOut
```

```
47  event ChangeBundleForSettleOrder(uint256 indexed bundleId, int256
        ↪ makerAmountTotal, int256 makerAmountRemaining);
```

```
57      int256 makerAmountRemaining,
        uint256 amountIn,
        uint128 takerFeeAmountIn
```

```
72      int256 amount0,
        int256 amount1,
```

```
88      uint256 amount0,
        uint256 amount1,
```

```
99  event Collect(address indexed sender, address indexed recipient,
        ↪ uint128 amount0, uint128 amount1);
```

```
106 event CollectFeeProtocol(address indexed sender, address indexed
        ↪ recipient, uint128 amount0, uint128 amount1);
```

```
113 event CollectFeeChannel(address indexed sender, address indexed
        ↪ recipient, uint128 amount0, uint128 amount1);
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** IGridParameters.sol

Description Here a signed type is used for a token amount, while in most of the other places unsigned types are used.

Recommendation Consider using the same type everywhere for token amounts.

Client Comment *The data type of this parameter needs to respect that of the amountSpecified data type. It represents the direction of the swap.*

```
38 int256 amountSpecifiedRemaining;
```

CVF-20. FIXED

- **Category** Suboptimal
- **Source** IGridStructs.sol

Recommendation As long as bundle IDs are assigned sequentially, smaller type could be used for them to save storage space. Using “uint64” for bundle IDs would save two storage slots per “Boundary” instance.

```
15 uint256 bundle0Id;
   uint256 bundle1Id;
```

CVF-21. FIXED

- **Category** Suboptimal
- **Source** IGridStructs.sol

Recommendation Using smaller type for boundary IDs could save one storage slot per “Order” instance.

```
21 uint256 bundleId;
```

7 Moderate Issues

CVF-22. INFO

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

Description Overflow is possible when converting to “uint128”.

Recommendation Consider using safe conversion.

Client Comment *The converted value will not overflow. The maximum amount of liquidity allowed in each boundary is less than or equal to uint128.*

```
55 uint256(-amountRemaining) > makerAmount ? makerAmount : uint128(
      ↪ uint256(-amountRemaining)),
```

CVF-23. INFO

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

Description Over-/underflow is possible here.

Recommendation Consider using checked math.

Client Comment *The business logic guarantees that it will not overflow.*

```
276 ? (priceCurrentX96 - priceDeltaX96WithRate).toUint160()
    : (priceCurrentX96 + priceDeltaX96WithRate).toUint160();
```

CVF-24. INFO

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

Description Underflow is possible when converting to “uint24”.

Recommendation Consider using safe conversion.

Client Comment *Paramater checks are guaranteed by GridFactory*

```
301 .mulDiv(uint24(takerFeePips), amountIn, 1e6 - uint24(takerFeePips),
      ↪ Math.Rounding.Up)
```

CVF-25. INFO

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

Description Underflow is possible when calculating the second parts.

Recommendation Consider using safe math.

Client Comment *The business logic guarantees that it will not overflow.*

```
315 ? (priceLimitX96 - priceCurrentX96, boundaryPriceX96 -
       ↪ priceCurrentX96)
    : (priceCurrentX96 - priceLimitX96, priceCurrentX96 -
       ↪ boundaryPriceX96);
```

CVF-26. FIXED

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

Description These constants are taken from Uniswap V3 TickMath, from a function that performs different calculations, so error estimations made for that function are not relevant for this function.

Recommendation Consider estimating the maximum errors for this function either experimentally or analytically, and calculating accurate constants for this function.

```
218 int24 boundaryLow = int24((log10001 -
       ↪ 340299295680913241859614010066024721O) >> 128);
    int24 boundaryHi = int24((log10001 +
       ↪ 291339464771989622907027621153398088495) >> 128);
```

CVF-27. INFO

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

Description Underflow is possible here.

Recommendation Consider using checked math.

Client Comment *The caller guarantees the parameter checks.*

```
235 boundaryLower = boundary - remainder;
    boundaryLower = remainder >= 0 ? boundaryLower : boundaryLower -
       ↪ resolution;
```

CVF-28. INFO

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

Description Overflow is possible here.

Recommendation Consider using checked math.

Client Comment *The caller guarantees the parameter checks.*

```
250 else if (boundaryLower + resolution > MAX_BOUNDARY) return
       ↪ boundaryLower - resolution;
```

CVF-29. INFO

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

Description This makes a swap to stop at a word boundary.

Recommendation Should probably be “continue” instead of “break”.

Client Comment *The function nextInitializedBoundary will search for the liquidity that has been initialized. If it cannot be found, it means that there is no available liquidity to swap*

```
273 if (!initialized) break;
```

CVF-30. FIXED

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

Description Over-/underflow is possible here.

Recommendation Consider using checked math.

Client Comment *New business logic: no longer taking in protocol fees.*

```
302 channelFees[channel].token0 = channelFees[channel].token0 +
       ↪ feeChannel;
    protocolFees.token0 = protocolFees.token0 + (state.feeProtocol -
       ↪ feeChannel);
```

```
305 channelFees[channel].token1 = channelFees[channel].token1 +
       ↪ feeChannel;
    protocolFees.token1 = protocolFees.token1 + (state.feeProtocol -
       ↪ feeChannel);
```

CVF-31. FIXED

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

Description Overflow is possible when converting “step.amountIn” to “int256”.

Recommendation Consider using safe conversion.

```
379 : state.amountSpecifiedRemaining - int256(step.amountIn) - int256(
       ↪ uint256(step.feeAmount));
```

CVF-32. FIXED

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

Description Overflow is possible here.

Recommendation Consider using checked math.

Client Comment *New business logic: no longer taking in protocol fees.*

```
441 state.feeProtocol = state.feeProtocol + takerFeeForProtocolAmount;
```

CVF-33. FIXED

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

Description Underflow is possible when converting to “uint24”.

Recommendation Consider using safe conversion.

Client Comment *New business logic: no longer taking in protocol fees.*

```
597 fee0 = Math.mulDiv(amount0, uint24(takerFee), 1e6, Math.Rounding.Up)
       ↪ .toUint128();
```

```
602 fee1 = Math.mulDiv(amount1, uint24(takerFee), 1e6, Math.Rounding.Up)
       ↪ .toUint128();
```

CVF-34. FIXED

- **Category** Documentation
- **Source** BundleMath.sol

Description Descriptions for these two returned values are identical. Most probably either of the descriptions is incorrect.

Recommendation Consider fixing the incorrect description.

```
83 /// @return makerAmountOut The amount of token0 or token1 that the
       ↪ maker will receive
   /// @return takerAmountOut The amount of token0 or token1 that the
       ↪ maker will receive
```

8 Minor Issues

CVF-35. INFO

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Description Signed word position looks weird.

Recommendation Consider using unsigned word positions.

```
13 wordPos = int16(boundary >> 8);
```

CVF-36. FIXED

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Recommendation This could be simplified as: bitPos = uint8 (uint24 (boundary));

```
14 bitPos = uint8(uint24(boundary % 256));
```

CVF-37. FIXED

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Recommendation This could be optimized as: uint256 mask = ~uint256(0) » ~bitPos;

```
102 uint256 mask = (1 << bitPos) - 1 + (1 << bitPos);
```

CVF-38. INFO

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Recommendation This could be simplified as: next = (wordPos « 8 | int24(uint24(initialized ? BitMath.mostSignificantBit(masked) : 0))) * resolution;

110
```
next = initialized
    ? (compressed - 1 - int24(uint24(bitPos - BitMath.
       ↪ mostSignificantBit(masked)))) * resolution
    : (compressed - 1 - int24(uint24(bitPos))) * resolution;
```

CVF-39. INFO

- **Category** Procedural
- **Source** BoundaryBitmap.sol

Description This looks like an attempt to fix a bug in the calling code.

Recommendation Consider moving this logic inthe the calling code.

114
```
if (boundary < 0 && boundary % resolution != 0) {
    // round towards negative infinity
    --compressed;
}
```

CVF-40. FIXED

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Recommendation This could be simplified as: uint256 mask = ~uint256(0) « bitPos;

122
```
uint256 mask = ~((1 << bitPos) - 1);
```

CVF-41. INFO

- **Category** Suboptimal
- **Source** BoundaryBitmap.sol

Recommendation "This could be simplified as: next = (wordPos « 8 | int24(uint24(initialized ? BitMath.leastSignificantBit(masked) : 255))) * resolution;"

```
130 next = initialized
        ? (compressed + 1 + int24(uint24(BitMath.leastSignificantBit(
           ↪ masked) - bitPos))) * resolution
        : (compressed + 1 + int24(uint24(type(uint8).max - bitPos))) *
           ↪ resolution;
```

CVF-42. FIXED

- **Category** Suboptimal
- **Source** SwapMath.sol

Description The expression "-amountRemaining' is calculated twice.

Recommendation Consider calculating once and reusing.

```
55 uint256(-amountRemaining) > makerAmount ? makerAmount : uint128(
      ↪ uint256(-amountRemaining)),
```

CVF-43. FIXED

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

Description Phantom overflow is possible when calculating -amountRemaining.

Recommendation Consider surrounding with "unchecked" block.

```
55 uint256(-amountRemaining) > makerAmount ? makerAmount : uint128(
      ↪ uint256(-amountRemaining)),
```

CVF-44. FIXED

- **Category** Procedural
- **Source** SwapMath.sol

Description The rest of the function looks like it is always executed while actually it is executed only when price is in range.

Recommendation Consider putting the rest of the function into an eplicit “else” branch.

```
78 }
```

CVF-45. INFO

- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation Consider doing safe casts here.

```
110 uint256 takerAmountInWithoutFee = Math.mulDiv(takerAmountInRemaining
    ↪ , 1e6 - uint256(uint24(takerFeePips)), 1e6);
```

CVF-46. INFO

- **Category** Procedural
- **Source** SwapMath.sol

Description Underflow is prevented here via a complicated business logic. Such approach is error-prone, as business logic may change in the future.

Recommendation Consider using checked math.

```
169 step.feeAmount = (takerAmountInRemaining - takerAmountInWithoutFee).
    ↪ toUint128();
```

CVF-47. INFO

- **Category** Flaw
- **Source** SwapMath.sol

Description There is no range check for this argument.

Recommendation Consider explicitly checking that it is not negative.

```
106 int24 takerFeePips
```

```
287 int24 takerFeePips
```

CVF-48. FIXED

- **Category** Suboptimal
- **Source** SwapMath.sol

Description Using the“mulDiv” function for dividing as 256-bit number by two is an overkill.

Recommendation Consider using right shift instead. Add 1 to the numerator before shifting for rounding up.

```
292 priceAvgX96 = Math.mulDiv(priceAccumulateX96, 1, 2,
    ↪ priceNextRounding).toUint160();
```

CVF-49. INFO

- **Category** Documentation
- **Source** SwapMath.sol

Description It actually cannot, as “a” has 192 bits and “b” has 96 bits at most.

```
330 // (a + b - 1) / b can overflow on addition, so we distribute
```

CVF-50. INFO

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

Description This function doesn't check boudary range, as it may mark out of range boundary as "valid".

Recommendation Consider renaming the function to emphasize that it only performs a resoludtion check.

```
17 function isValidBoundary(int24 boundary, int24 resolution) internal
   ↪ pure returns (bool isValid) {
```

CVF-51. INFO

- **Category** Flaw
- **Source** BoundaryMath.sol

Description There are no range checks for the "resolution" arguments.

Recommendation Consider adding appropriate checks.

```
17 function isValidBoundary(int24 boundary, int24 resolution) internal
   ↪ pure returns (bool isValid) {
```

```
232 function getBoundaryLowerAtBoundary(int24 boundary, int24 resolution
   ↪ ) internal pure returns (int24 boundaryLower) {
```

```
246     int24 resolution
```

CVF-52. FIXED

- **Category** Suboptimal
- **Source** BoundaryMath.sol

Recommendation The "else" part could be simplified as: uint24(boundary).

```
41 uint256 absBoundary = boundary < 0 ? uint256(-int256(boundary)) :
   ↪ uint256(int256(boundary));
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** BoundaryMath.sol

Recommendation This could be simplified as: priceX96 = uint160(ratio + 0xFFFFFFFF » 32);

```
72 priceX96 = uint160((ratio >> 32) + (ratio % (1 << 32) == 0 ? 0 : 1))
   ↪ ;
```

CVF-54. FIXED

- **Category** Suboptimal
- **Source** BoundaryMath.sol

Recommendation This function could be simplified as: return boundary - (boundary % resolution + resolution) % resolution.

```
232 function getBoundaryLowerAtBoundary(int24 boundary, int24 resolution
    ↪ ) internal pure returns (int24 boundaryLower) {
```

CVF-55. FIXED

- **Category** Procedural
- **Source** BitMath.sol

Description Despite the pragma, this code is not compatible with Solidity versions prior to 0.8.0.

Recommendation Consider changing the pragma to "^0.8.0".

```
2  pragma solidity >=0.5.0;
```

```
16         unchecked {
```

```
61         unchecked {
```

CVF-56. INFO

- **Category** Suboptimal
- **Source** BitMath.sol

Recommendation This could be simplified and optimized as: uint256 y; if ((y = x » 128) != 0) { x = y; r += 128; } ... if ((y = x » 2) != 0) { x = y; r += 2; } if (x » 1 != 0) { r += 1; }

Client Comment *The contract size will increase, which may not necessarily optimize gas consumption.*

```
17  if (x >= 0x100000000000000000000000000000000) {
        x >>= 128;
        r += 128;
20  }
    if (x >= 0x10000000000000000) {
        x >>= 64;
        r += 64;
    }
    if (x >= 0x100000000) {
        x >>= 32;
        r += 32;
    }
    if (x >= 0x10000) {
30      x >>= 16;
        r += 16;
    }
    if (x >= 0x100) {
        x >>= 8;
        r += 8;
    }
    if (x >= 0x10) {
        x >>= 4;
        r += 4;
40  }
    if (x >= 0x4) {
        x >>= 2;
        r += 2;
    }
    if (x >= 0x2) {
        r += 1;
    }
```

CVF-57. INFO

- **Category** Suboptimal
- **Source** BitMath.sol

Recommendation This function could be simplified as: unchecked { return mostSignificantBit (x & uint256 (-int256 (x))); }

```
57 function leastSignificantBit(uint256 x) internal pure returns (uint8
   ↪  r) {
```

CVF-58. INFO

- **Category** Procedural
- **Source** TradingConfig.sol

Description This version requirement is inconsistent with version requirements in other files.

Recommendation Consider specifying as ‘^0.8.0”.

```
2 pragma solidity ^0.8.8;
```

CVF-59. FIXED

- **Category** Documentation
- **Source** TradingConfig.sol

Description The number format for this value is unclear.

Recommendation Consider documenting.

```
9 int24 constant ALLOW_MAX_FEE = 1e4;
```

CVF-60. INFO

- **Category** Suboptimal
- **Source** TradingConfig.sol

Recommendation The initial fees should be passed as constructor arguments.

```
15 fees[1] = FeeConfig({takerFee: 100, makerFee: -80});
   emit ResolutionEnabled(1, 100, -80);
```

```
18 fees[5] = FeeConfig({takerFee: 500, makerFee: -400});
   emit ResolutionEnabled(5, 500, -400);
```

```
21 fees[30] = FeeConfig({takerFee: 3000, makerFee: -2400});
   emit ResolutionEnabled(30, 3000, -2400);
```

CVF-61. INFO

- **Category** Suboptimal
- **Source** TradingConfig.sol

Recommendation This check is redundant as it is anyway possible to pass a dead collector address.

```
64 require(newCollector != address(0), "TC_PFCZ");
```

CVF-62. INFO

- **Category** Procedural
- **Source** GridFactory.sol

Description This version requirement is inconsistent with version requirements in other files.

Recommendation Consider specifying as ‘^0.8.0”.

```
2 pragma solidity ^0.8.8;
```

CVF-63. INFO

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

Recommendation The type of this variable should be “ITradingConfig”.

```
12 address public immutable override tradingConfig;
```

CVF-64. INFO

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

Recommendation The type of this variable should be “IPriceOracle”.

```
13 address public immutable override priceOracle;
```

CVF-65. INFO

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

Recommendation The type of this variable should be “IWETH9”.

```
14 address public immutable weth9;
```

CVF-66. FIXED

- **Category** Documentation
- **Source** GridFactory.sol

Description The semantics of the keys and values in this mapping is unclear.

Recommendation Consider documenting.

```
15 mapping(address => mapping(address => mapping(int24 => address)))
     ↪ public override grids;
```

CVF-67. INFO

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

Recommendation The type of this variable should be: mapping(IERC20 ⇒ mapping(IERC20 ⇒ mapping(int24 ⇒ IGrid)))

```
15 mapping(address => mapping(address => mapping(int24 => address)))
      ↪ public override grids;
```

CVF-68. INFO

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

Recommendation The type of this "_tradingConfig" argument should be "ITradingConfig".

```
17 constructor(address _tradingConfig, address _weth9, bytes memory
      ↪ _gridCreationCode) {
```

CVF-69. INFO

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

Recommendation The type of the "_weth9" argument should be "IWETH9".

```
17 constructor(address _tradingConfig, address _weth9, bytes memory
      ↪ _gridCreationCode) {
```

CVF-70. INFO

- **Category** Suboptimal
- **Source** GridFactory.sol

Description These checks are redundant. It is anyway possible to pass addresses of wrong contracts or destroy contracts after deploying the grid factory.

Recommendation Consider removing these checks.

```
19 require(Address.isContract(_tradingConfig), "GF_NC");
20 require(Address.isContract(_weth9), "GF_NC");
```

```
32 require(Address.isContract(tokenA), "GF_NC");
   require(Address.isContract(tokenB), "GF_NC");
```

CVF-71. INFO

- **Category** Procedural
- **Source** GridDeployer.sol

Description This version requirement is inconsistent with version requirements in other files.

Recommendation Consider specifying as ‘^0.8.0”.

```
2 pragma solidity ^0.8.8;
```

CVF-72. INFO

- **Category** Suboptimal
- **Source** GridDeployer.sol

Description Storing a creation code in the storage costs much gas. It seems that the code is stored only once in the constructor.

Recommendation Consider storing the creation code in the byte code of the GridFactory contract using “type(Grid).creationCode”.

Client Comment *Will cause the contract size to be too large to deploy.*

```
13 gridCreationCode = creationCode;
```

CVF-73. INFO

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

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

```
22 address token0,
   address token1,
```

CVF-74. INFO

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

Recommendation The type of this argument should ne “ITradingConfig”.

```
25 address tradingConfig,
```

CVF-75. INFO

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

Recommendation The type of this argument should be “IPriceOracle”.

```
26 address priceOracle,
```

CVF-76. INFO

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

Recommendation The type of this argument should be “IWETH9”.

```
27 address weth9
```

CVF-77. INFO

- **Category** Suboptimal
- **Source** GridDeployer.sol

Recommendation This could be done in a type safe way: grid = new Grid {salt: keccak256(abi.encode(token0, token1, resolution))} ();

```
30 grid = Create2.deploy(0, keccak256(abi.encode(token0, token1,
      ↪ resolution)), gridCreationCode);
```

CVF-78. INFO

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

Recommendation The type of this variable should be “IGridFactory”.

```
10 address public immutable gridFactory;
```

CVF-79. INFO

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

Recommendation The key type for these mappings should be “IGrid”.

```
12 mapping(address => GridOracleState) public override gridOracleStates
      ↪ ;
   mapping(address => GridPriceData[65535]) public override
      ↪ gridPriceData;
```

CVF-80. INFO

- **Category** Suboptimal
- **Source** PriceOracle.sol

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

```
12 mapping(address => GridOracleState) public override gridOracleStates
       ↪ ;
   mapping(address => GridPriceData[65535]) public override
       ↪ gridPriceData;
```

CVF-81. INFO

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

Recommendation The type of token arguments should be “IERC20”.

```
20 function register(address tokenA, address tokenB, int24 resolution)
       ↪ external override {
```

CVF-82. INFO

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

Recommendation The type of the “grid” argument should be “IGrid”.

```
28 function _register(address grid) internal {
```

```
46 function _update(address grid, int24 boundary, uint32 blockTimestamp
       ↪ ) internal {
```

```
72 function increaseCapacity(address grid, uint16 capacityNext)
       ↪ external override {
```

```
94     address grid,
```

```
108    address grid,
```

CVF-83. INFO

- **Category** Documentation
- **Source** PriceOracle.sol

Description It is unclear what overflows are meant here, as there are several places inside the block where overflow is possible.

Recommendation Consider explaining.

```
53 // safe for 0 or 1 overflows
```

CVF-84. INFO

- **Category** Suboptimal
- **Source** PriceOracle.sol

Recommendation The “% state.capacity” part is redundant.

```
233 uint256 left = (state.index + 1) % state.capacity;
```

CVF-85. INFO

- **Category** Procedural
- **Source** Grid.sol

Description This version requirement is inconsistent with version requirements in other files.

Recommendation Consider specifying as ‘^0.8.0”.

```
2 pragma solidity ^0.8.8;
```

CVF-86. INFO

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

Recommendation The type of these variables should be “IERC20”.

```
35 address public immutable override token0;
   address public immutable override token1;
```

CVF-87. INFO

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

Recommendation The type of this variable should be “IWETH9”.

```
39 address private immutable weth9;
```

CVF-88. INFO

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

Recommendation The type of this variable should be “ITradingConfig”.

```
40 address private immutable tradingConfig;
```

CVF-89. INFO

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

Recommendation The type of this variable should be “IPriceOracle”.

```
41 address private immutable priceOracle;
```

CVF-90. INFO

- **Category** Documentation
- **Source** Grid.sol

Description The semantics of the keys in these mappings is Unclear.

Recommendation Consider documenting.

```
50 mapping(int24 => Boundary) public override boundaries0;
   mapping(int24 => Boundary) public override boundaries1;
```

CVF-91. INFO

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

Description The names are confusing.

Recommendation Consider renaming to “_nextOrderId” and “_nextBundleId”.

```
55 uint256 private _orderId;
```

```
58 uint256 private _bundleId;
```

CVF-92. INFO

- **Category** Procedural
- **Source** Grid.sol

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

```
64 receive() external payable {}
```

CVF-93. FIXED

- **Category** Suboptimal
- **Source** Grid.sol

Recommendation Both values, boundaryLower and boundaryLower + resolution, are checked agains both range ends, while it would be enough to check boundaryLower against the lower range end, and boundaryLower + resolution agains the upper range end.

```
151 BoundaryMath.isInRange(boundaryLower) &&
        BoundaryMath.isInRange(boundaryLower + resolution) &&
```

CVF-94. FIXED

- **Category** Suboptimal
- **Source** Grid.sol

Recommendation Using “WithAmounts” version of the “addLiquidity” function is redundant here, as newly allocated bundles are guaranteed to be empty.

```
174 bundle.addLiquidityWithAmount(0, 0, amount);
```

```
190 bundle.addLiquidityWithAmount(makerAmountTotal, makerAmountRemaining
    ↪ , amount);
```

CVF-95. INFO

- **Category** Suboptimal
- **Source** Grid.sol

Recommendation It would be more reasonable to compare block numbers rather than block timestamps.

```
287 slot0Cache.blockTimestamp != (blockTimestamp = uint32(block.
    ↪ timestamp))
```

CVF-96. FIXED

- **Category** Suboptimal
- **Source** Grid.sol

Recommendation Unary minus would be more efficient.

```
396     SafeCast.toInt256(parameters.amountOutUsed) * -1,
```

```
414             SafeCast.toInt256(parameters.amountOutUsed) * -1,
```

```
457 ? (token1, token0, SafeCast.toInt256(amountInputTotal), SafeCast.
    ↪ toInt256(amountOutputTotal) * -1)
: (token0, token1, SafeCast.toInt256(amountOutputTotal) * -1,
    ↪ SafeCast.toInt256(amountInputTotal));
```

```
545 int256(uint256(order.amount)) * -1,
int256(uint256(makerAmountOut)) * -1
```

CVF-97. INFO

- **Category** Suboptimal
- **Source** Grid.sol

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

```
476 if (amount0 > 0) tokensOwed.token0 = tokensOwed.token0 + amount0;
    if (amount1 > 0) tokensOwed.token1 = tokensOwed.token1 + amount1;
```

```
616     protocolFees.token0 = protocolFees.token0 + paid0;
```

```
624     protocolFees.token1 = protocolFees.token1 + paid1;
```

CVF-98. FIXED

- **Category** Procedural
- **Source** Grid.sol

Description These code blocks are very similar and differ only in what token is being sent out.

Recommendation Consider extracting into a function.

```
570 if (unwrapWETH9 && token0 == weth9) {
        IWETHMinimum(weth9).withdraw(amount0);
        Address.sendValue(payable(recipient), amount0);
    } else {
        SafeERC20.safeTransfer(IERC20(token0), recipient, amount0);
    }
```

```
578 if (unwrapWETH9 && token1 == weth9) {
        IWETHMinimum(weth9).withdraw(amount1);
580     Address.sendValue(payable(recipient), amount1);
    } else {
        SafeERC20.safeTransfer(IERC20(token1), recipient, amount1);
    }
```

CVF-99. INFO

- **Category** Suboptimal
- **Source** Grid.sol

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

```
673 tokensOwed.token0 = tokensOwed.token0 - amount0;
```

```
681 tokensOwed.token1 = tokensOwed.token1 - amount1;
```

CVF-100. INFO

- **Category** Suboptimal
- **Source** BundleMath.sol

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

```
40 self.takerAmountRemaining = self.takerAmountRemaining + (parameters.
   ↪ amountInUsed).toUint128();
```

```
50 self.takerFeeAmountRemaining = self.takerFeeAmountRemaining +
   ↪ parameters.takerFeeForMakerAmountUsed;
```

```
57 self.makerAmountTotal = self.makerAmountTotal + makerAmount;
```

```
59     self.makerAmountRemaining = self.makerAmountRemaining +
       ↪ makerAmount;
```

CVF-101. INFO

- **Category** Suboptimal
- **Source** BundleMath.sol

Description Overflow here is prevented implicitly by the previous assignment. Such implicit stuff is error-prone and makes code harder to read.

Recommendation Consider using checked math.

```
59 self.makerAmountRemaining = self.makerAmountRemaining + makerAmount;
```

```
76 self.makerAmountRemaining = makerAmountRemaining + makerAmount;
```

CVF-102. INFO

- **Category** Procedural
- **Source** FixedPointX192.sol

Description These two constants have the same value.

Recommendation Consider merging them into one.

```
5 uint256 internal constant RESOLUTION = 1 << 192;
  uint256 internal constant Q = 0
     ↪ x100000000000000000000000000000000000000000000000;
```

CVF-103. INFO

- **Category** Suboptimal
- **Source** GridAddress.sol

Description Hardcoding this value is error-prone.

Recommendation Consider using “keccak256(type(Grid).creationCode)” instad.

```
7 bytes32 internal constant GRID_BYTES_CODE_HASH = 0
     ↪ xa96e546c9f84f633bde3489a08d8935cfd9546d7ce1b5369a6df136a69afc
  7af;
```

CVF-104. INFO

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

Recommendation The type of these fields should be “IERC20”.

```
10 address token0;
   address token1;
```

CVF-105. INFO

- **Category** Procedural
- **Source** FixedPointX96.sol

Description These two constants have the same value.

Recommendation Consider merging them into one.

```
5 uint160 internal constant RESOLUTION = 1 << 96;
  uint160 internal constant Q = 0x1000000000000000000000000;
```

CVF-106. FIXED

- **Category** Suboptimal
- **Source** FeeMath.sol

Description The “mulDiv” function is redundant here as all arguments are 128 bits.

Recommendation Consider using plain 256-bits multiplication and division.

```
22 takerFeeForMakerAmount = Math.mulDiv(makerFeePipsAbs, takerFeeAmount
     ↪ , uint24(takerFeePips)).toUint128();
```

CVF-107. FIXED

- **Category** Procedural
- **Source** FeeMath.sol

Description This statement does nothing.

Recommendation Consider removing it.

```
26 return (takerFeeForMakerAmount, takerFeeForProtocolAmount);
```

CVF-108. INFO

- **Category** Procedural
- **Source** FixedPointX128.sol

Description These two constants have the same value.

Recommendation Consider merging into one.

```
5 uint160 internal constant RESOLUTION = 1 << 128;
  uint160 internal constant Q = 0x100000000000000000000000000000000;
```

CVF-109. INFO

- **Category** Procedural
- **Source** IWETHMinimum.sol

Description This interface seems to contain an arbitrary subset of the “IWETH9” interface.

Recommendation Consider using the original “IWETH9” interface instead.

```
4 interface IWETHMinimum {
```

CVF-110. INFO

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

Recommendation Events are usually named via nouns, such as “Resolution” and “ResolutionUpdate”.

```
10 event ResolutionEnabled(int24 indexed resolution, int24 indexed
     ↪ takerFee, int24 indexed makerFee);
```

```
16 event ResolutionUpdated(int24 indexed resolution, int24 indexed
     ↪ newTakerFee, int24 indexed newMakerFee);
```

```
22 event ProtocolFeeCollectorTransferred(
```

CVF-111. INFO

- **Category** Procedural
- **Source** ITradingConfig.sol

Recommendation These two events are very similar and could be merged into one event.

```
10 event ResolutionEnabled(int24 indexed resolution, int24 indexed
      ↪ takerFee, int24 indexed makerFee);
```

```
16 event ResolutionUpdated(int24 indexed resolution, int24 indexed
      ↪ newTakerFee, int24 indexed newMakerFee);
```

CVF-112. INFO

- **Category** Suboptimal
- **Source** ITradingConfig.sol

Recommendation The parameter is redundant as its value could be derived from the previous event.

```
24 address indexed oldCollector,
```

CVF-113. INFO

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

Description This function should return a single value of type “FeeConfig”.

```
40 function fees(int24 resolution) external view returns (int24
      ↪ takerFee, int24 makerFee);
```

CVF-114. INFO

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

Description The word “transfer” is confusing.

Recommendation Consider naming as “setProtocolFeeCollector”.

```
63 function transferProtocolFeeCollector(address newCollector) external
   ↪ ;
```

CVF-115. INFO

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

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

```
10 event IncreaseCapacity(address indexed grid, uint16 capacityOld,
   ↪ uint16 capacityNew);
```

CVF-116. INFO

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

Recommendation The type of the “grid” parameter should be “IGrid”.

```
10 event IncreaseCapacity(address indexed grid, uint16 capacityOld,
   ↪ uint16 capacityNew);
```

CVF-117. INFO

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

Recommendation The type of the “grid” argument” should be “IGrid”.

```
35 function gridOracleStates(address grid) external view returns (
   ↪ uint16 index, uint16 capacity, uint16 capacityNext);
```

```
44     address grid,
```

```
62 function increaseCapacity(address grid, uint16 capacityNext)
   ↪ external;
```

```
68 function getBoundaryCumulative(address grid, uint32 secondsAgo)
   ↪ external view returns (int56 boundaryCumulative);
```

```
75     address grid,
```

CVF-118. INFO

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

Recommendation The type of the token arguments should be “IERC20”.

```
52 function register(address tokenA, address tokenB, int24 resolution)
   ↪ external;
```

CVF-119. INFO

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

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

```
11 event GridCreated(address indexed token0, address indexed token1,
   ↪ int24 indexed resolution, address grid);
```

CVF-120. INFO

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

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

```
11 event GridCreated(address indexed token0, address indexed token1,
   ↪ int24 indexed resolution, address grid);
```

CVF-121. INFO

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

Recommendation The type of the “grid” parameter should be “IGrid”.

```
11 event GridCreated(address indexed token0, address indexed token1,
   ↪ int24 indexed resolution, address grid);
```

CVF-122. INFO

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

Recommendation The return type should be “ITradingConfig”.

```
14 function tradingConfig() external view returns (address);
```

CVF-123. INFO

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

Recommendation The return type should be “IPriceOracle”.

```
17 function priceOracle() external view returns (address);
```

CVF-124. INFO

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

Recommendation The type of the token arguments should be “IERC20”.

```
25 function grids(address tokenA, address tokenB, int24 resolution)
     ↪ external view returns (address grid);
```

```
33 function createGrid(address tokenA, address tokenB, int24 resolution
     ↪ ) external returns (address grid);
```

CVF-125. INFO

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

Recommendation The return type should be “IGrid”.

```
25 function grids(address tokenA, address tokenB, int24 resolution)
     ↪ external view returns (address grid);
```

```
33 function createGrid(address tokenA, address tokenB, int24 resolution
     ↪ ) external returns (address grid);
```

CVF-126. INFO

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

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

Recommendation Consider renaming.

```
25 function grids(address tokenA, address tokenB, int24 resolution)
     ↪ external view returns (address grid);
```

CVF-127. INFO

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

Recommendation Events are usually named via nouns, such as “MakerOrder” or “MakerOrderSettlement”.

```
20 event PlaceMakerOrder(
```

```
34 event SettleMakerOrder(
```

```
47 event ChangeBundleForSettleOrder(uint256 indexed bundleId, int256
   ↪ makerAmountTotal, int256 makerAmountRemaining);
```

```
55 event ChangeBundleForSwap(
```

```
106 event CollectFeeProtocol(address indexed sender, address indexed
   ↪ recipient, uint128 amount0, uint128 amount1);
```

```
113 event CollectFeeChannel(address indexed sender, address indexed
   ↪ recipient, uint128 amount0, uint128 amount1);
```

CVF-128. INFO

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

Recommendation The type of these fields should be “IERC20”.

```
10 address token0;
address token1;
```

CVF-129. INFO

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

Recommendation The type of this field should be “ITradingConfig”.

```
13 address tradingConfig;
```

CVF-130. INFO

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

Recommendation The type of this field should be “IPriceOracle”.

```
14 address priceOracle;
```

CVF-131. INFO

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

Recommendation The type of this field should be “IWETH9”.

```
15 address weth9;
```

CVF-132. INFO

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

Description This function should return a single value of type “Parameters”.

```
29 function parameters()
```

CVF-133. INFO

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

Recommendation The type of these returned values should be “IERC20”.

```
33 address token0,
   address token1,
```

CVF-134. INFO

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

Recommendation The type of this returned value should be “ITradingConfig”.

```
36 address tradingConfig,
```

CVF-135. INFO

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

Recommendation The type of this returned value should be “IPriceOracle”.

```
37 address priceOracle,
```

CVF-136. INFO

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

Recommendation The type of this returned value should be “IWETH9”.

```
38 address weth9
```

CVF-137. INFO

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

Recommendation The returned type should be “IERC20”.

```
12 function token0() external view returns (address);
```

```
15 function token1() external view returns (address);
```

CVF-138. INFO

- **Category** Suboptimal
- **Source** IGrid.sol

Recommendation Consider using “bytes32” as the return type to emphasize, that the returned value is not a number but rather a bit mask.

```
55 function boundaryBitmaps0(int16 wordPos) external view returns (
       ↪ uint256 word);
```

```
58 function boundaryBitmaps1(int16 wordPos) external view returns (
       ↪ uint256 word);
```

CVF-139. INFO

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

Recommendation The type of this returned value should be a enum with two values.

```
88 bool zero,
```

CVF-140. INFO

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

Description These functions should emit some events and these events should be declared in this interface.

123
```
function swap(
```

134
```
function placeMakerOrder(
```

141
```
function placeMakerOrderInBatch(
```

150
```
function settleMakerOrder(uint256 orderId) external returns (uint128
    ↪  amount0, uint128 amount1);
```

157
```
function settleMakerOrderAndCollect(
```

168
```
function settleMakerOrderAndCollectInBatch(
```

181
```
function flash(address recipient, uint256 amount0, uint256 amount1,
    ↪ bytes calldata data) external;
```

191
```
function collect(
```

204
```
function collectProtocolFees(
```

217
```
function collectChannelFees(
```

CVF-141. INFO

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

Recommendation The type of this argument should be a enum with two values.

126
```
bool zeroForOne,
```

CVF-142. FIXED

- **Category** Documentation
- **Source** IGrid.sol

Description This argument is not documented.

Recommendation Consider documenting.

```
171 bool unwrapWETH9
```

CVF-143. INFO

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

Recommendation The type of this field should be a enum with two values: “TOKEN_0” and “TOKEN_1”.

```
21 bool zero;
```

CVF-144. INFO

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

Recommendation The type of this field should be a enum with two values.

```
35 bool zeroForOne;
```

CVF-145. FIXED

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

Description The semantics of this field is unclear.

Recommendation Consider using a more descriptive name or adding a documentation comment.

```
56 bool stopSwap;
```

CVF-146. INFO

- **Category** Procedural
- **Source** IGridParameters.sol

Description Some amounts are represented as “uint256” while other are represented as “uint128”.

Recommendation Consider using the same type for all amounts.

```
74 uint256 amountInUsed;
76 uint256 amountInRemaining;
78 uint128 amountOutUsed;
80 uint128 amountOutRemaining;
82 uint128 takerFeeForMakerAmountUsed;
84 uint128 takerFeeForMakerAmountRemaining;
```

CVF-147. INFO

- **Category** Procedural
- **Source** IGridSwapCallback.sol

Description This callback uses the “int256” type for token amounts, while other Gridex callbacks use “uint256” and “uint128”.

Recommendation Consider using the same type for token amounts in all callbacks for consistency.

```
1 // SPDX-License-Identifier: GPL-2.0-or-later
```

CVF-148. INFO

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

Description Word “grid” in the interface name but “gridex” in the function name.

Recommendation Consider using consistent naming.

```
6  interface IGridSwapCallback {
```

```
16     function gridexSwapCallback(int256 amount0Delta, int256
          ↪ amount1Delta, bytes calldata data) external;
```

CVF-149. INFO

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

Description Word “grid” in the interface name but “gridex” in the function name.

Recommendation Consider using consistent naming.

```
6  interface IGridPlaceMakerOrderCallback {
```

```
16     function gridexPlaceMakerOrderCallback(uint256 amount0, uint256
          ↪ amount1, bytes calldata data) external;
```

CVF-150. INFO

- **Category** Procedural
- **Source** IGridPlaceMakerOrderCallback.sol

Description This callback uses the “uint256” type for token amounts, while other Gridex callbacks use “uint128” and “int256”.

Recommendation Consider using the same type for token amounts in all callbacks for consistency.

```
16 function gridexPlaceMakerOrderCallback(uint256 amount0, uint256
      ↪ amount1, bytes calldata data) external;
```

CVF-151. INFO

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

Description Word “grid” in the interface name but “gridex” in the function name.

Recommendation Consider using consistent naming.

```
6  interface IGridFlashCallback {
```

```
13     function gridexFlashCallback(uint128 fee0, uint128 fee1, bytes
          ↪ calldata data) external;
```

CVF-152. INFO

- **Category** Procedural
- **Source** IGridFlashCallback.sol

Description This callback uses the “uint128” type for token amounts, while other Gridex callbacks use “uint256” and “int256”.

Recommendation Consider using the same type for token amounts in all callbacks for consistency.

```
13 function gridexFlashCallback(uint128 fee0, uint128 fee1, bytes
      ↪ calldata data) external;
```

CVF-153. INFO

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

Description The semantics of this field is unclear.

Recommendation Consider using a more specific name or explaining in a documentation comment.

```
7  bool zero;
```

CVF-154. INFO

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

Recommendation The type of this field should be a enum with two values: “TOKEN_0” and “TOKEN_1”.

```
7 bool zero;
```

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