Report

v. 2.0

Customer

Equation

Smart contract Audit

Equation DAO V2

4th February 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	03.02.24	A. Zveryanskaya	Initial Draft
0.2	03.02.24	A. Zveryanskaya	Minor revision
1.0	03.02.24	A. Zveryanskaya	Release
1.1	04.02.24	A. Zveryanskaya	CVF-6,7 downgraded
2.0	04.02.24	A. Zveryanskaya	Release

2 Introduction

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

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	MarketManager.sol	MarketManager States.sol
core/interfaces		
IConfigurable.sol	IMarketErrors.sol	IMarketLiquidity Position.sol
IMarketManager.sol	IMarketPosition.sol	
libraries/		
ConfigurableUtil.sol	Constants.sol	FundingRateUtil.sol
LiquidityPositionUtil.sol	MarketUtil.sol	Math.sol
PositionUtil.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 12 moderate, and a few less important issues.

	Info	Fixed
Moderate	10	2
Minor	15	2

Fixed 4 out of 29 issues

6 Moderate Issues

CVF-1. INFO

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

Description Using a business-level field for a low-level purpose of identifying whether a market is already enabled is a bad practice. Zero max leverage may become a valid value in the future, making this check invalid.

Recommendation Consider using a separate flag for detecting whether a market is enabled.

Client Comment *Storing fewer values can save gas.*

```
13 +if (_self[_market].baseConfig.maxLeveragePerLiquidityPosition > 0)
```

```
31 +if (marketCfg.baseConfig.maxLeveragePerLiquidityPosition == 0)
      ↪ revert IConfigurable.MarketNotEnabled(_market);
```

```
46 +if (marketCfg.baseConfig.maxLeveragePerLiquidityPosition == 0)
      ↪ revert IConfigurable.MarketNotEnabled(_market);
```

```
61 +if (marketCfg.baseConfig.maxLeveragePerLiquidityPosition == 0)
      ↪ revert IConfigurable.MarketNotEnabled(_market);
```

CVF-2. INFO

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

Description The “Math” library isn’t imported.

Recommendation Consider importing it.

Client Comment *Math has been imported through another file.*

```
178 +int256 unrealizedPnLGrowthDeltaX64 = Math
```

```
220 +uint256 maintenanceMargin = Math.ceilDiv(
```

```
242 +? Math.mulDiv(uint256(unrealizedPnLGrowthDeltaX64), _positionCache.
       ↪ liquidity, Constants.Q64).toInt256()
243 +: -Math.mulDivUp(uint256(-unrealizedPnLGrowthDeltaX64),
       ↪ _positionCache.liquidity, Constants.Q64).toInt256();
```

CVF-3. INFO

- **Category** Bad naming
- **Source** PositionUtil.sol (ONLY YELLOW)

Description The name looks like a “view” function, while actually this function does modify the storage.

Recommendation Consider renaming.

```
791 +function _calculateFee(
```

CVF-4. INFO

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

Description Using a business-level field for a low-level purpose of identifying whether a market is already enabled is a bad practice. Zero max leverage may become a valid value in the future, making this check invalid.

Recommendation Consider using a separate flag for detecting whether a market is enabled.

Client Comment *Storing fewer values can save gas.*

```
116 +function _isEnabledMarket(IMarketDescriptor _market) internal view
    ↪ returns (bool) {
117 +    return marketConfigs[_market].baseConfig.
    ↪ maxLeveragePerLiquidityPosition != 0;
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** MarketManagerStates.sol

Description The storage address for “reentrancyStates[_market]” is calculated several times.

Recommendation Consider refactoring to calculate only once.

```
20 +if (reentrancyStatus[_market] == ENTERED) revert ReentrancyGuard.
   ↪ ReentrancyGuardReentrantCall();
```

```
22 +reentrancyStatus[_market] = ENTERED;
```

```
24 +reentrancyStatus[_market] = NOT_ENTERED;
```

CVF-6. INFO

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

Description As “balanceAfter” could be greater than “usdBalance + _amount”, any extra balance remains unaccounted and could be claimed by subsequent transfers.

Recommendation Consider setting “usdBalance” to the value of “balanceAfter” to ensure all the balance is accounted.

Client Comment *Excessive balance will not be accepted by the protocol.*

```
352 +usdBalance += _amount;
353 +_state.usdBalance += _amount;
```

CVF-7. INFO

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

Description There is no explicit check to ensure that “usdBalance” is sufficient for the transfer. In case it is not, underflow will happen and transaction will be terminated without any meaningful error message.

Recommendation Consider explicitly checking the balance and reverting with an error message in case the balance is insufficient.

```
357 +usdBalance -= _amount;
358 +_state.usdBalance -= _amount;
```

CVF-8. INFO

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

Description Overflow is possible here.

Recommendation Consider using checked math.

Client Comment *There won’t be an overflow here.*

```
221 +    uint256(_liquidity) * _baseCfg.
    ↪ liquidationFeeRatePerLiquidityPosition,
```

```
223 +) + _baseCfg.liquidationExecutionFee;
```

CVF-9. INFO

- **Category** Suboptimal
- **Source** MarketUtil.sol

Description It is unclear why the statement in the comment is true. There is nothing in the surrounding code to guarantee it. Even if it is enforced by some code located elsewhere, such approach is very error-prone as it introduces hidden relationships between code parts.

Recommendation Consider using checked math to avoid overflow.

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

```
50 +// Because `positionLiquidityAfter` is less than or equal to `
      ↪ globalLiquidityAfter`, it will not overflow
51 +uint256 positionLiquidityAfter = _state.liquidationFundPositions[
      ↪ _account] + _liquidityDelta;
```

CVF-10. INFO

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

Description Underflow is possible here.

Recommendation Consider using checked math.

Client Comment *There will be no overflow here; necessary checks have been performed above.*

```
78 +_state.globalLiquidationFund.liquidity -= _liquidityDelta;
79 +_state.globalLiquidationFund.liquidationFund -= int256(uint256(
      ↪ _liquidityDelta));
```

```
81 +uint256 positionLiquidityAfter = positionLiquidity -
      ↪ _liquidityDelta;
```

CVF-11. FIXED

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

Description Overflow here will cause transaction termination due to failed assert. If overflow is indeed desired, consider surrounding this line with an "unchecked" block.

```
205 +uint128 totalNetSize = netSize + liquidationBufferNetSize; //
      ↪ overflow is desired
```

CVF-12. FIXED

- **Category** Documentation
- **Source** FundingRateUtil.sol (ONLY YELLOW)

Description The formula in the comment doesn't match the code, as the code has "liquidity" in denominator.

Recommendation Consider making the code consistent with the comment.

```
240 +// PnLGrowthDelta = (paidSize - receivedSize) / paidSize * (
        ↪ paidSize * price * fundingRate)
241 +//                 = (paidSize - receivedSize) * price * fundingRate
242 +//                 = (paidSize - receivedSize) *
        ↪ paidFundingRateGrowthDelta
```

```
245 +    unrealizedPnLGrowthDeltaX64 = Math
246 +        .mulDiv(
247 +            paidSize - receivedSize,
248 +            uint192(paidFundingRateGrowthDeltaX96),
249 +            Constants.Q32 * liquidity
250 +        )
251 +        .toInt256();
```

7 Minor Issues

CVF-14. FIXED

- **Category** Procedural
- **Source** PositionUtil.sol (ONLY YELLOW)

Recommendation Brackets are redundant.

```
831 +liquidityFee -= (referralFee + referralParentFee);
```

CVF-15. INFO

- **Category** Procedural
- **Source** MarketUtil.sol

Description We didn't review this file.

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

```
6 +import "../IEquationContractsV1Minimum.sol";
```

CVF-16. INFO

- **Category** Procedural
- **Source** MarketUtil.sol

Description The "Math" library isn't imported.

Recommendation Consider importing it.

Client Comment *Math has been imported through another file.*

```
122 +Math.min(_state.globalLiquidityPosition.liquidity, _marketPriceCfg.
    ↪ maxPriceImpactLiquidity)
```

```
220 +    isSameSign ? Math.Rounding.Down : Math.Rounding.Up
```

```
270 +size = Math.mulDiv(balanceRateX96, _liquidity, _indexPriceX96).
    ↪ toUint128();
```

CVF-18. INFO

- **Category** Procedural
- **Source** MarketUtil.sol

Recommendation This commented out line should be removed.

```
214 +// abs(priceDeltaX96) * totalNetSize / (liquidity * (1 << 32))
```

CVF-19. INFO

- **Category** Procedural
- **Source** FundingRateUtil.sol (ONLY YELLOW)

Description This version requirement is inconsistent with other file. Also, specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for: MarketManager.sol.

Client Comment *Consider specifying as "^0.8.0" unless there is something special about this particular version.*

```
2 -pragma solidity ^0.8.0;
2 +pragma solidity =0.8.23;
```

CVF-21. INFO

- **Category** Procedural
- **Source** IConfigurable.sol

Description We didn't review this file.

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

```
4 +import "./IMarketDescriptor.sol";
```

CVF-22. FIXED

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

Description The number format for these fields is unclear.

Recommendation Consider documenting.

```
19 +uint64 minMarginPerLiquidityPosition;
```

```
21 +uint32 maxLeveragePerLiquidityPosition;
```

```
27 +uint64 minMarginPerPosition;
```

```
29 +uint32 maxLeveragePerPosition;
```

CVF-23. INFO

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

Recommendation Events are usually named via nouns, such as “MarketConfig” or “MarketBaseConfig”.

```
102 +event MarketConfigEnabled(
```

```
112 +event MarketBaseConfigChanged(IMarketDescriptor indexed market,
    ↪ MarketBaseConfig newCfg);
```

```
117 +event MarketFeeRateConfigChanged(IMarketDescriptor indexed market,
    ↪ MarketFeeRateConfig newCfg);
```

```
122 +event MarketPriceConfigChanged(IMarketDescriptor indexed market,
    ↪ MarketPriceConfig newCfg);
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** IMarketErrors.sol

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

```
44 +error MaxPremiumRateExceeded();
```

```
48 +error LiquidationFundLoss();
```

CVF-25. INFO

- **Category** Procedural
- **Source** IMarketLiquidityPosition.sol

Description We didn't review this file.

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

```
4  +import "./IMarketDescriptor.sol";
```

CVF-26. INFO

- **Category** Documentation
- **Source** IMarketLiquidityPosition.sol

Description The number format for this field is unclear.

Recommendation Consider documenting.

```
33 +uint128 margin;
```

CVF-27. INFO

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

Recommendation Events are usually named via nouns, such as "GlobalLiquidityPosition-NetPositionChange" or "LiquidityPositionIncrease".

```
46  +event GlobalLiquidityPositionNetPositionChanged(
60  +event LiquidityPositionIncreased(
77  +event LiquidityPositionDecreased(
96  +event LiquidityPositionLiquidated(
108 +event PreviousSPPriceInitialized(IMarketDescriptor indexed market,
      ↪ uint160 previousSPPriceX96);
121 +event SettlementPointReached(
130 +event GlobalLiquidityPositionPnLGrowthIncreasedByFundingFee(
139 +event GlobalLiquidityPositionPnLGrowthIncreasedByTradingFee(
```

CVF-28. INFO

- **Category** Procedural
- **Source** IMarketPosition.sol

Description We didn't review this file.

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

```
4  +import "./IMarketDescriptor.sol";
```

CVF-29. INFO

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

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

```
74  +event GlobalFundingRateSampleAdjusted(
```

```
84  +event GlobalPositionSizeChanged(
```

CVF-30. INFO

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

Recommendation The number of vertices should be a named constant.

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

```
28  +PriceVertex[10] priceVertices;
```

```
30  +uint128[10] liquidationBufferNetSizes;
```

CVF-31. INFO

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

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

```
86  +event PriceVertexChanged(
```

```
96  +event ProtocolFeeIncreased(IMarketDescriptor indexed market,
        ↪ uint128 amount);
```

```
101 +event ProtocolFeeCollected(IMarketDescriptor indexed market,
        ↪ uint128 amount);
```

```
110 +event ReferralFeeIncreased(
```

```
124 +event ReferralFeeCollected(
```

```
134 +event PriceFeedChanged(IPriceFeed indexed priceFeedBefore,
        ↪ IPriceFeed indexed priceFeedAfter);
```

```
139 +event PremiumRateChanged(IMarketDescriptor indexed market, uint128
        ↪ premiumRateAfterX96);
```

```
145 +event LiquidationBufferNetSizeChanged(IMarketDescriptor indexed
        ↪ market, uint8 index, uint128 netSizeAfter);
```

```
150 +event BasisIndexPriceChanged(IMarketDescriptor indexed market,
        ↪ uint160 basisIndexPriceAfterX96);
```

```
156 +event GlobalLiquidationFundGovUsed(
```

```
171 +event GlobalLiquidationFundIncreasedByLiquidation(
```

```
181 +event LiquidationFundPositionIncreased(
```

```
192 +event LiquidationFundPositionDecreased(
```

CVF-32. INFO

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

Description We didn't review this file.

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

```
6  4  import "../governance/Governable.sol";
```

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