Report

v. 2.0

Customer

Exactly

Smart Contract Audit

Protocol update

14th February 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	12.01.24	A. Zveryanskaya	Initial Draft
0.2	12.01.24	A. Zveryanskaya	Minor revision
1.0	12.01.24	A. Zveryanskaya	Release
1.1	14.02.24	A. Zveryanskaya	Client comments have been added for all issues
1.2	14.02.24	A. Zveryanskaya	Links to the original and fixed code have been updated
2.0	14.02.24	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.

Exactly is a decentralized, non-custodial and open-source protocol that provides an autonomous fixed and variable interest rate market enabling users to frictionlessly exchange the time value of their assets and completing the DeFi credit market.

3 Project scope

We were asked to review the protocol updates:

- Original Code
- Code with Fixes

Files:

/			
	Auditor.sol	InterestRateModel.sol	Market.sol
	MarketETHRouter.sol	PriceFeedDouble.sol	PriceFeedPool.sol
	PriceFeedWrapper.sol	RewardsController.sol	
utils/			
	FixedLib.sol		
periphery/			
	EXA.sol	Airdrop.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 1 major, and a few less important issues. All identified Major issues have been fixed.

	Info	Fixed
Major	0	1
Moderate	0	3
Minor	7	3

Fixed 7 out of 14 issues

6 Major Issues

CVF-1. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider using the ”mulDiv” function or prevent phantom overflow is some other way.

Client Comment *We improved the constructor checks when calling ‘fixedRate‘ and ‘baseRate‘ using extreme values to avoid an overflow caused by misconfigurations. Additionally, we are sharing the study of each of the 3 situations with realistic but extreme market conditions. The results are shared on a separate note to simplify this document.*

142
```
+v.natPools = ((2e18 - int256(v.fNatPools)) * 1e36) / (int256(v.
   ↪ auxFNatPools) * (1e18 - int256(v.auxFNatPools)));
```

154
```
+          ((v.natPools * int256((v.fixedFactor * 1e18).sqrt())) /
+            1e18 +
+            ((1e18 - v.natPools) * int256(v.fixedFactor)) /
+            1e18 -
```

185
```
+                (-((sigmoidSpeed * (int256(uGlobal.divWadDown(1e18
   ↪ - uGlobal)).lnWad() - auxUNat)) / 1e18)).expWad()
```

7 Moderate Issues

CVF-2. FIXED

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

Description Overflow is possible when converting to ”int256”.

Recommendation Consider using safe conversion.

Client Comment *We adopted SafeCast lib from OpenZeppelin.*

```
97  +auxUNat = int256(floatingNaturalUtilization.divWadDown(
        ↪ fixedNaturalUtilization)).lnWad();
```

```
142 +v.natPools = ((2e18 - int256(v.fNatPools)) * 1e36) / (int256(v.
        ↪ auxFNatPools) * (1e18 - int256(v.auxFNatPools)));
```

```
146 +    (((int256(maturitySpeed) *
    +      int256(
```

```
154 +          ((v.natPools * int256((v.fixedFactor * 1e18).sqrt())) /
```

```
156 +            ((1e18 - v.natPools) * int256(v.fixedFactor)) /
```

```
172 +int256 r = int256(floatingCurveA.divWadDown(floatingMaxUtilization
        ↪ - uFloating)) + floatingCurveB;
```

```
179 +    int256(
```

```
185 +                (-((sigmoidSpeed * (int256(uGlobal.divWadDown(1e18
        ↪ - uGlobal)).lnWad() - auxUNat)) / 1e18)).expWad()
```

```
280 +      -int256(
```

CVF-3. FIXED

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

Description Underflow is possible when converting to ”uint256”.

Recommendation Consider using safe conversion.

Client Comment *We adopted SafeCast lib from OpenZeppelin.*

```
144 +uint256 spread = uint256(
```

```
177 +uint256 globalFactor = uint256(
```

```
277 +uint256 averageFactor = uint256(
```

CVF-4. FIXED

- **Category** Flaw
- **Source** InterestRateModel.sol

Description This formula throws in case floatingAssets < floatingDebt + backupBorrowed.

Recommendation Consider returning zero in such a case or reverting with a meaningful error message.

Client Comment *We rewrote the formula in a different way to avoid this case and handle utilizations bigger than 1e18.*

```
346 +return floatingAssets != 0 ? 1e18 - (floatingAssets - floatingDebt
    ↪ - backupBorrowed).divWadDown(floatingAssets) : 0;
```

8 Minor Issues

CVF-5. INFO

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

Recommendation The value 365 days should be a named constant.

Client Comment *This is a design choice.*

```
931 +.mulDivDown(block.timestamp - lastFloatingDebtUpdate, 365 days)
```

```
956 +.mulDivDown(block.timestamp - lastFloatingDebtUpdate, 365 days)
```

CVF-6. FIXED

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

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

Client Comment *This value was simplified when rewriting the formula.*

```
1053 +return assets > 0 ? 1e18 - (assets - debt - backupBorrowed).
       ↪ divWadDown(assets) : 0;
```

CVF-7. FIXED

- **Category** Flaw
- **Source** Market.sol

Description This formula throws in case assets < debt + backupBorrowed.

Recommendation Consider returning zero in such a case or reverting with a meaningful error message.

Client Comment *We rewrote the formula in a different way to avoid this case and handle utilizations bigger than 1e18.*

```
1053 +return assets > 0 ? 1e18 - (assets - debt - backupBorrowed).
       ↪ divWadDown(assets) : 0;
```

CVF-8. INFO

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

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

Client Comment *This is a design choice.*

```
554 +? 1e18 - (m.floatingAssets - m.floatingDebt - market.
       ↪ floatingBackupBorrowed()).divWadDown(m.floatingAssets)
```

```
568 +      (int256(v.globalUtilization.divWadDown(1e18 - v.
       ↪ globalUtilization)).lnWad() -
```

```
582 +  .mulWadDown(1e18 - v.globalUtilization.mulWadUp(1e18 - market.
       ↪ treasuryFeeRate())) +
```

CVF-9. FIXED

- **Category** Documentation
- **Source** InterestRateModel.sol

Description The number format for these variables is unclear.

Recommendation Consider documenting.

Client Comment *We added natspec.*

```
32 +uint256 public immutable floatingNaturalUtilization;
   +uint256 public immutable fixedNaturalUtilization;
   +int256 public immutable growthSpeed;
   +int256 public immutable sigmoidSpeed;
   +uint256 public immutable maxRate;
   +int256 public immutable spreadFactor;
```

```
39 +uint256 public immutable maturitySpeed;
```

CVF-10. INFO

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

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

Client Comment *This is a design choice.*

```
67 +    maxUtilization_ > 1e18 &&

69 +    floatingNaturalUtilization_ < 1e18 &&

95 +fixedNaturalUtilization = 1e18 - floatingNaturalUtilization;

139 +v.fNatPools = (maxPools * 1e18).divWadDown(fixedNaturalUtilization)
       ↪ ;
140 +v.auxFNatPools = (v.fNatPools * 1e18).sqrt();

142 +v.natPools = ((2e18 - int256(v.fNatPools)) * 1e36) / (int256(v.
       ↪ auxFNatPools) * (1e18 - int256(v.auxFNatPools)));

145 +  1e18 +

151 +      ).lnWad()) / 1e18).expWad() *

154 +          ((v.natPools * int256((v.fixedFactor * 1e18).sqrt())) /
    +            1e18 +
    +            ((1e18 - v.natPools) * int256(v.fixedFactor)) /
    +            1e18 -
    +            1e18)) /
    +        1e18)) /
160 +    1e18

170 +if (uGlobal >= 1e18) return type(uint256).max;

180 +      1e18 -
    +        uint256(1e18)

184 +               1e18 +
    +                 (-((sigmoidSpeed * (int256(uGlobal.divWadDown(1e18
       ↪ - uGlobal)).lnWad() - auxUNat)) / 1e18)).expWad()

189 +    ).lnWad()) / 1e18).expWad()
```

(278, 287, 346)

CVF-11. INFO

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

Recommendation The value ”15_000e6” should be a named constant.

Client Comment *This is a design choice.*

```
73 +maxRate_ <= 15_000e16 &&
```

CVF-12. INFO

- **Category** Suboptimal
- **Source** InterestRateModel.sol

Description Rendering this value in such way looks very odd, as the value is in WAD format.

Recommendation Consider rendering as ”150e18”.

Client Comment *This is a design choice for percentage notations.*

```
73 +maxRate_ <= 15_000e16 &&
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** InterestRateModel.sol

Recommendation This conversion is redundant.

Client Comment *‘divWadDown‘ cannot be called directly on a literal.*

```
181 +uint256(1e18)
```

CVF-14. INFO

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

Recommendation The value ”365 days” should be a named constant.

Client Comment *This is a design choice.*

```
270 -return fixedRate(utilizationBefore, utilizationAfter).mulDivDown(
       ↪ maturity - block.timestamp, 365 days);
```

```
316 +    ).mulDivDown(block.timestamp - lastFloatingDebtUpdate, 365 days
       ↪ )
```

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