Report

v. 1.0

Customer

Exactly

Smart Contract Audit

Protocol Update

27th March 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	25.03.25	A. Zveryanskaya	Initial Draft
0.2	26.03.25	A. Zveryanskaya	Minor revision
1.0	27.03.25	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 diff covering the protocol update and the corresponding code fixes.

Files:

```
/
        Market.sol                  RewardController.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

Several recommendations were provided to the Client.

	Info	Fixed
Moderate	1	1
Minor	5	2

6 Moderate Issues

CVF-1. INFO

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

Description Underflow is possible here.

Recommendation Either properly support situation when the fixed principals value is less than the fixed consolidated value, or clearly explain why such situation is not possible.

Client Comment *As soon as the Market and RewardsController contracts are upgraded, the Exactly team will call the 'initConsolidated' function for each account that interacted with fixed operations. During this process, which may take a few minutes, the contracts will remain unpaused, meaning accounts can still interact with them. If an account operates before its initialization is complete, the new 'market.initConsolidated(account);' hook in the RewardsController will automatically initialize the variables for that account doing the work for us. This hook is triggered before any subtractions where an underflow could have occurred, ensuring safe execution.*

```
1175 +fixedOps.borrows += borrows - fixedConsolidated[account].borrows;
```

```
1182 +fixedOps.deposits += deposits - fixedConsolidated[account].deposits
        ↪ ;
```

CVF-2. FIXED

- **Category** Suboptimal
- **Source** Market.sol

Description Performing this check on each iteration of the loop is inefficient.

Recommendation Perform the check once before the loop and store a reference to the proper mapping in a "storage" variable.

```
1201 +FixedLib.Position memory position = isBorrow
1202 +  ? fixedBorrowPositions[maturity][account]
1203 +  : fixedDepositPositions[maturity][account];
```

7 Minor Issues

CVF-3. INFO

- **Category** Suboptimal
- **Source** Market.sol

Recommendation It would be more efficient to merge these two maps into a single map whose keys are account addresses, and values are structs, encapsulating the values of the original maps.

Client Comment *Once all accounts are initialized with their fixed consolidated borrows and deposits, the 'isInitialized' mapping (address ⇒ bool) and the 'initConsolidated' function will be removed in a future contract upgrade. For this reason, we prefer not to merge the mappings, even if it offers a short-term improvement.*

```
97  +mapping(address account => FixedOps consolidated) public
       ↪ fixedConsolidated;
```

```
101 +mapping(address account => bool initialized) public isInitialized;
```

CVF-4. INFO

- **Category** Procedural
- **Source** Market.sol

Recommendation When a boolean literal is passed as a function argument, it is a good practice to put the argument name as a comment next to the literal, like this: handleRewards(true /* isBorrow */, borrower);

Client Comment *We haven't followed this practice before, and adopting it now would disorder with our code consistency/standards.*

```
155 +handleRewards(true, borrower);
215 +handleRewards(true, borrower);
254 +handleRewards(false, receiver);
307 +handleRewards(true, borrower);
390 +handleRewards(false, owner);
511 +handleRewards(true, borrower);
661 +      handleRewards(true, borrower);
724 +handleRewards(false, borrower);
766 +handleRewards(false, owner);
779 +handleRewards(false, owner);
785 +handleRewards(false, to);
795 +handleRewards(false, msg.sender);
796 +handleRewards(false, to);
807 +handleRewards(false, from);
808 +handleRewards(false, to);
1181 +  handleRewards(false, account);
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** Market.sol

Description The value “position.principal” is read from the storage twice.

Recommendation Read once and reuse.

```
677 +fixedConsolidated[borrower].borrows -= position.principal;
678 +fixedOps.borrows -= position.principal;
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** Market.sol

Recommendation Maturities could be packed even tighter by using the following optimizations: 1. Divide the ”maturity” field value by “FixedLib.INTERVAL” before packing. This allows using less bits for “maturity”: 12 instead of 32. 2. Consider the maturity referenced by the “maturity” fields to always be selected, so the bit mask starts from the next maturity. This will save 1 more bit.

Client Comment *This seems like a great enhancement.*
However, we are unsure how such a change would work with already stored maturities values, it would probably require a migration. Additionally, implementing this would likely require multiple changes across all functions where we handle packed maturities. Given these factors, this doesn’t seem like a simple refinement that can be thoroughly assessed within the scope of the current audit.

```
1197 +uint256 maturity = packedMaturities & ((1 << 32) - 1);
1198 +packedMaturities = packedMaturities >> 32;
```

CVF-7. INFO

- **Category** Procedural
- **Source** RewardsController.sol

Description The storage address of “dist.availableRewards” is calculated on every loop iteration.

Recommendation Calculate once before the loop.

Client Comment *Given the complexity of caching intermediary storage slots, we prefer to avoid these kind of optimizations.*

```
90  +update(account, Market(msg.sender), dist.availableRewards[r], ops);
```

```
176 178  if (claimedAmount > 0) {
```

CVF-8. INFO

- **Category** Procedural
- **Source** RewardsController.sol

Recommendation The value “rewardList[r]” should be calculated outside the inner loop.

Client Comment *Unfortunately, due to the 'Stack too deep' compiler error, this recommendation isn't feasible. While we could explore ways to fit the stack, doing so would reduce readability and at this stage increase the risk of bugs if the code changes significantly. For these reasons, we prefer to keep the loop as it is.*

```
300 +RewardData storage rewardData = dist.rewards[rewardList[r]];
```

CVF-9. FIXED

- **Category** Procedural
- **Source** RewardsController.sol

Recommendation This check should be performed before calculating “rewardData”.

```
301 +if (dist.availableRewardsCount == 0) {
```

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