Report

v. 1.0

Customer

Exactly

Smart Contract Audit

Protocol update. Part II

15th April 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	15.04.24	A. Zveryanskaya	Initial Draft
0.2	15.04.24	A. Zveryanskaya	Minor revision
1.0	15.04.24	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

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 changes for the Market.sol and the new periphery/InstallmentsRouter.sol contract :

- Original Code
- Code with Fixes

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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Recommendations

We found several minor issues and provided Client with recommendations for consideration.

CVF-1 INFO

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

Description In ERC-20 the name and symbol properties are usually meant immutable. Modifying them could cause problems with certain software.

Client Comment *We are aware that external integrations can have issues related to caching. We plan to contact third parties to notify the change. For example, etherscan.*

```
1060 +name = string.concat("exactly␣", assetSymbol);
     +symbol = string.concat("exa", assetSymbol);
```

CVF-3 FIXED

- **Category** Procedural
- **Source** InstallmentsRouter.sol

Recommendation Consider specifying as ”^0.8.0” unless there is something special regarding this particular version.

Client Comment *We updated the range to ”^0.8.0”.*

```
2 +pragma solidity ^0.8.17;
```

CVF-4 INFO

- **Category** Procedural
- **Source** InstallmentsRouter.sol

Description We didn’t review this file.

```
4 +import { WETH, SafeTransferLib } from "solmate/src/tokens/WETH.sol"
    ↪ ;
```

CVF-5 FIXED

- **Category** Suboptimal
- **Source** InstallmentsRouter.sol

Recommendation This expression could be simplified by using a counter initially set to ”firstMaturity” and increased by ”FixedLib.INTERVAL” on each iteration.

Client Comment *We applied the simplification.*

```
65 +firstMaturity + i * FixedLib.INTERVAL,
```

CVF-6 FIXED

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

Description It is unclear what ”checks” means here.

Recommendation Consider emphasizing that the function reverts in case the check failed.

Client Comment *We improved natspec.*

```
125 +/// @notice Checks if the Market is listed by the Auditor.
```

CVF-7 FIXED

- **Category** Procedural
- **Source** InstallmentsRouter.sol

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

Client Comment *We improved natspec.*

```
146 +{} catch {} // solhint-disable-line no-empty-blocks
```

CVF-8 INFO

- **Category** Suboptimal
- **Source** InstallmentsRouter.sol

Description Silently ignoring failed transactions is a bad practice.

Recommendation Consider emitting an event with the error.

Client Comment *This is a version of a known pattern to avoid DoS: https://www.trust-ecurity.xyz/post/permission-denied https://github.com/trust1995/trustlessPermit/blob/-main/TrustlessPermit.sol.*

```
146 +{} catch {} // solhint-disable-line no-empty-blocks
```

CVF-9 INFO

- **Category** Suboptimal
- **Source** InstallmentsRouter.sol

Description Declaring top-level types and errors in a file named after a contract makes code harder to navigate.

Recommendation Consider either moving the declarations into the contract or moving them into a separate file.

Client Comment *This is a design choice.*

```
151 +struct Permit {
```

```
159 +error Disagreement();
160 +error NotFromWETH();
```

CVF-10 INFO

- **Category** Suboptimal
- **Source** InstallmentsRouter.sol

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

Client Comment *These error signatures are already part of the exactly protocol interface, so this is a design choice.*

```
159 +error Disagreement();
160 +error NotFromWETH();
```

CVF-11 INFO

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

Description This function should emit some event.

Client Comment *As we intend to use this function only for USDC.e and native USDC compatibility, this function will be removed to free some bytecode space, and adding an event would complicate backward compatibility.*

1059
```
+function setAssetSymbol(string calldata assetSymbol) public
   ↪ onlyRole(DEFAULT_ADMIN_ROLE) {
```

CVF-12 FIXED

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

Description This event is emitted even if nothing actually changed.

Client Comment *We changed it.*

1156
```
+emit Frozen(msg.sender, isFrozen_);
```

CVF-13 FIXED

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

Recommendation The ”account” parameter should be indexed.

Client Comment *We changed it.*

1362
```
+event Frozen(address account, bool isFrozen);
```

CVF-14 INFO

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

Description Declaring top-level errors in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the declarations into the contract or moving them into a separate file.

Client Comment *This is a design choice.*

1377
```
+error MarketFrozen();
```

1379
```
+error NotPausingRole();
```

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