Report

v. 1.0

Customer

Uniswap

Smart Contract Audit

UniswapX v.1.1

27th July 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	28.07.23	A. Zveryanskaya	Initial Draft
0.2	28.07.23	A. Zveryanskaya	Minor revision
1.0	28.07.23	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.

The Uniswap Protocol is an open-source protocol for providing liquidity and trading ERC20 tokens on Ethereum. It eliminates trusted intermediaries and unnecessary forms of rent extraction, allowing for safe, accessible, and efficient exchange activity. The protocol is non-upgradable and designed to be censorship resistant.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

interfaces/			
	IReactor.sol	IReactorCallback.sol	
lens/			
	OrderQuoter.sol		
lib/			
	CurrencyLibrary.sol	ExpectedBalanceLib.sol	FillDataLib.sol
reactors/			
	BaseReactor.sol		
sample-executors/			
	SwapRouter02 Executor.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 3 major, and a few less important issues. All identified Major issues have been fixed.

	Active	Fixed
Major	0	3
Moderate	0	1
Minor	0	2

Fixed 6 out of 6 issues

6 Major Issues

CVF-2. FIXED

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

Description Using “fillData” to control whether callback ought to be executed makes API messy and more error-prone.

Recommendation Consider using a separate boolean flag.

Client Comment *Switched to separate _withCallback_ entrypoints.*

```
73 +if (fillData.executeReactorCallback()) {
```

CVF-4. FIXED

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

Description The function name is very confusing, as one would thing that it actually does execute a reactor callback, while it just tells, whether such callback ought to be executed.

Recommendation Consider renaming.

Client Comment *Removed.*

```
11 +function executeReactorCallback(bytes calldata fillData) internal
   ↪ pure returns (bool) {
```

CVF-5. FIXED

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

Description This check makes reactor behavior quite unpredictible, as single zero byte could be a normal fill data.

Recommendation Consider using another way that would allow arbitrary fill data to be used, or at least would use a not so collision-prone special fill data to mark skipped callbacks.

Client Comment *Removed.*

```
12 +return fillData.length != 1 || fillData[0] != SKIP_REACTOR_CALLBACK
   ↪ ;
```

7 Moderate Issues

CVF-6. FIXED

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

Description Without the “filler” argument it would be harder for a callback to recognise malicious calls.

Recommendation Consider explaining how to verify calls without this argument.

```
13 -function reactorCallback(ResolvedOrder[] memory resolvedOrders,
      ↪ address filler, bytes memory fillData) external;
12 +function reactorCallback(ResolvedOrder[] memory resolvedOrders,
      ↪ bytes memory fillData) external;
```

8 Minor Issues

CVF-1. FIXED

- **Category** Procedural
- **Source** SwapRouter02Executor.sol

Description These checks could be done via a modifier.

```
40 +if (msg.sender != whitelistedCaller) {
41 +    revert CallerNotWhitelisted();
42 +}
```

```
49 +if (msg.sender != whitelistedCaller) {
50 +    revert CallerNotWhitelisted();
51 +}
```

CVF-3. FIXED

- **Category** Documentation
- **Source** CurrencyLibrary.sol

Description It is unclear from this comment why is the filler.

Recommendation Consider calling him “caller” or “message sender”.

```
31 +/// @notice Transfer currency from the filler to recipient
```

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