1inch Audit

Fixed Rate Swap

August 2021

By CoinFabrik

Introduction

CoinFabrik was asked to audit the FixedRateSwap contract for 1inch. FixedRateSwap is a simple Automated Market Maker (AMM) that allows swapping tokens with a 1:1 rate and a variable fee. The fee is calculated depending on the ratio of balances of the tokens in the contract by documented formulae. Deposits are limited to onlyOnwer by design.

First we will provide a summary of our discoveries and then we will show the details of our findings.

Summary

The contracts audited are deployed and can be checked at the following link (https://etherscan.io/address/0x40bbdE0eC6F177C4A67360d0f0969Cfc464b0bB4#code).

Contracts

The audited contract is:

- `FixedRateSwap.sol`

The contract imports standard libraries from OpenZeppelin which have been considered as part of the audit.

Analyses

The following analyses were performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks

- Functions with excessive gas cost
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Moreover, formulas were validated so that there are no special cases in which a user could take advantage of these calculations or drain the contract's balance; and that code implements formulas from the documentation correctly.

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

No issues found in this category.

Medium severity

No issues found in this category.

Minor severity

MI-01 Division by Zero in getReturn()

The function `getReturn()` is a public function that is called by `_swap()` and the swap methods to compute swap fees. If `inputAmount` is zero, there is a division by zero that should be avoided.

Recommendation

Require inputAmount to be positive.

MI-02 Division by Zero in depositFor()

The function `depositFor()` is a public function that can only be called by the owner. At one point the function computes `totalBalance` as the sum of the contract's balances of tokens 0 and 1, and then computes the variable share dividing a value by this number. No checks are made to ensure this value is nonzero, and hence there is a chance of division by zero which should be avoided.

Recommendation

Require `token0Amount` to be smaller than the contract's balance of token0 and `token1Amount` to be smaller than the contract's balance of token1.

Observations

Imprecise Computations and Constants

There is an error of a few digits in the constants C2 and C3 that could be improved. While the impact of using the original approximations to C2 and C3 as opposed to the ones computed herein is typically small, we suggest using those we computed

We repeat the logic leading to the computation of these constants.

First let

$$f(x) = 0.9999 + (a \cdot x + b)^k$$

such that f(0) = 1 and f(1) = 0.998.

Since f(0) = 0.9999 + (a . 0 + b)^k = 0.9999 + b^k= 1, then

$$b = (0.0001)^{1/17}$$

The constant b can be approximated as 581709132937435757 (representing the number b/1e18).

Similarly, since 0.998=f(1) = 0.9999+(a+b)^k, we have that 0.998 = 0.9999 + (a + b)^k or equivalently

$$a = -0.0019^{1/k} - b$$

Hence, the constant a can be better approximated as a =-127342331881541970 which represents the number (a/1e18). Say, with these choices of ***a*** and ***b***, we have (a + b)18 = 1900000000000000 / 1e18 whereas with the original values the result is 1899999999999998 /1e18 that is slightly worse.

Now, the fee getReturn(x) for an input amount x is defined as

$$\begin{aligned} \textbf{getReturn}(x) := & \quad \frac{1}{|I|}\int_I f(x)\,dx \\ = & \quad \frac{c(x_1-x_0)+\frac{1}{a\cdot(k+1)}(a\cdot x+b)^{k+1}\Big|_{x_0}^{x_1}}{x_1-x_0} \end{aligned}$$

where integration is done over the interval I=[x_0,x_1]=

[
inputBalance/(inputBalance + outputBalance),
(inputBalance + inputAmount) / (inputBalance + outputBalance)
].

So that, if

$$\begin{aligned}\textbf{getReturn}(x) :=& \int_I f(x)\,dx \\ =& \frac{C_1(x_1-x_0)+C_2|x_0-C_3|^{k+1}-C_2|x_1-C_3|^{k+1}}{x_1-x_0}\end{aligned}$$

we must have

$$\begin{aligned}-C_2(x_0-C_3)^{k+1} =& \frac{1}{(k+1)\cdot a}\left(a\cdot x_0+b\right)^{k+1} \\ =& \frac{a^{k+1}}{(k+1)\cdot a}\left(\frac{a\cdot x_0}{a}+b\right)^{k+1} \\ =& \frac{a^{k+1}}{(k+1)\cdot a}\left(x_0-\frac{b}{-a}\right)^{k+1} \\ =& \frac{a^k}{k+1}\left(x_0-\frac{b}{-a}\right)^{k+1}\end{aligned}$$

and hence

$$\begin{aligned}C_1 =& \ c \\ C_2 =& \ -\frac{a^k}{k+1} \\ C_3 =& \ -\frac{b}{a}\end{aligned}$$

We can now compute

C1 = 999900000000000000,

C2 = 3382712334998325432, and

C3 = 456807350974663119

Conclusion

We found the contracts to be simple, straightforward and documentation to be sufficient. A trivial issue was found in getReturn with a division by zero, although the security impact is minimal. We further checked the mathematics for the getReturn() functions and suggested better approximations to the constants.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the 1inch project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.