SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Rubicon Finance
Prepared by: Sherlock
Lead Security Expert: mstpr-brainbot
Dates Audited: February 5 - February 8, 2024
Prepared on: February 21, 2024

Introduction

Trading infrastructure for the internet.

Scope

Repository: RubiconDeFi/gladius-contracts-internal

Branch: master

Commit: ef95e2500c3f836479a7679eb7617812ccceaaa3

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
1	0

Issues not fixed or acknowledged

Medium	High
0	0

Issue M-1: Pairs with "MAX_FEE" can revert due to rounding inconsistencies

Source: https://github.com/sherlock-audit/2024-02-rubicon-finance-judging/issues/51

Found by

KingNFT, Kow, cawfree, mstpr-brainbot

Summary

If the pair has set the max fee by the fee controller admin which is "1_000" then depending on the amount to be swapped, the tx can revert due to rounding error.

Vulnerability Detail

When the fee amount is calculated inside the RubiconFeeController, the fee amount is rounded down.

```
uint256 feeAmount = fee.applyFee
                ? order.outputs[i].amount.mulDivUp(fee.fee, DENOM)
                : order.outputs[i].amount.mulDivUp(baseFee, DENOM);
```

then, ProtocolFees abstract contract will do a double check on the fee taken as follows:

```
if (feeOutput.amount > tokenValue.mulDivDown(MAX_FEE, DENOM)) {
                revert FeeTooLarge(
                    feeOutput.token,
                    feeOutput.amount,
                    feeOutput.recipient
                );
            }
```

As we can see, it uses mulDivDown, so if the calculation in the FeeController rounds up, the transaction will revert.

Textual PoC: Suppose the fee pair is set to "1_000" for tokens A and B. Alice sends an order to sell "11111111111111111111" (111.11 in 18 decimals) token A for token B.

Within the fee controller, the fee amount will be calculated as: 111111111111111111111 * 1000 / 100_000 (roundUp) = 111111111111111112

Subsequently, during execution, within the ProtocolFees contract, the maximum fee amount will be computed as: 111111111111111111111 * 1000 / 100_000 (roundDown) = 111111111111111111

Consequently, the transaction will revert because 111111111111111111 > 111111111111111112.

Coded PoC:

```
// forge test --match-contract GladiusReactorTest --match-test test_FeesRounding
↪  -vv
    function test_FeesRounding(uint amount) external {
        // @dev there will be plenty of values reverting this test.
        vm.assume(amount <= type(uint128).max);
        vm.assume(amount >= 1e6);

        uint DENOM = 100_000;
        uint FEE = 1_000;

        uint resultDown = FixedPointMathLib.mulDivDown(amount, FEE, DENOM);
        uint resultUp = FixedPointMathLib.mulDivUp(amount, FEE, DENOM);

        assertEq(resultDown, resultUp);
    }
```

Impact

As stated in README: **Fee Controller Can DOS Trading Activity. Note, that, as said above, the resulting output shouldn't overflow MAX_FEE, but other possibilities of reverts are known/acceptable.** Any fee setting in range 0<MAX_FEE should not revert and if it reverts then its acceptable. Hence, I'll label this as medium.

Code Snippet

https://github.com/sherlock-audit/2024-02-rubicon-finance/blob/11cac67919e8a1303b3a3177291b88c0c70bf03b/gladius-contracts-internal/src/fee-controllers/RubiconFeeController.sol#L63-L116

https://github.com/sherlock-audit/2024-02-rubicon-finance/blob/11cac67919e8a1303b3a3177291b88c0c70bf03b/gladius-contracts-internal/src/base/ProtocolFees.sol#L39-L105

Tool used

Manual Review

Recommendation

Discussion

sherlock-admin

1 comment(s) were left on this issue during the judging contest.

0xAadi commented:

sherlock-admin2

> Escalate
>
> Invalid this already a known issue reported in the open zeppelin audit of uniswapx https://github.com/Uniswap/UniswapX/blob/main/audit/v1.1/OpenZeppelin.pdf Look for L-02
>
> from contest readme
>
> > Please list any known issues/acceptable risks that should not result in a valid finding. From OZ audit - https://github.com/Uniswap/UniswapX/blob/main/audit/v1.1/OpenZeppelin.pdf
>
> The uniswapX decided to keep this issue as wont fix.
>
> also from rubicon readme
>
> > GladiusReactor is based on UniswapX's ExclusiveDutchOrderReactor and intended to support ExclusiveDutchOrders.
>
> Changes in fee calculation may result in support for ExclusiveDutchOrders not working. Therefore this should be set as wont fix and also invalid.

You've deleted an escalation for this issue.

daoio

the issue is valid, because the audit doc specifically states that while a DoS possibility from a fee-controller is a known and acceptable issue, the possibility of the `MAX_FEE` overflow isn't acceptable. Moreover, it not only notes a likelihood of a rounding error, but the root of its occurency - an incosistency between calculations in `RubiconFeeController` and `ProtocolFees`, where *validation* of the fee calculation is performed with a different function call (`mulDivDown` instead of `mulDivUp`)

ArnieGod

@daoio you are right, oversight on my part will remove escalation.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/RubiconDeFi/gladius-contracts-internal/pull/16.

sherlock-admin

The Lead Senior Watson signed-off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.