SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	Alchemix
Prepared by:	Sherlock
Lead Security Expert:	xiaoming90
Dates Audited:	April 15 - April 19, 2024
Prepared on:	June 5, 2024

Introduction

Alchemix is a self repaying loans protocol. This contest covers the synthetic xerc20-enabled assets and the reward router that enables incentives for loan-takers.

Scope

Repository: alchemix-finance/v2-foundry

Branch: reward-collector-fix

Commit: 9e01b533cc3eca2aaf2a6cb6b78b3077fae9c7d3

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
0	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

jasonxiale

Bauer

zigtur

ge6a

Issue H-1: The calculated value for slippage protection in the protocol is inaccurate

Source: https://github.com/sherlock-audit/2024-04-alchemix-judging/issues/5

Found by

Bauer, ge6a, jasonxiale, zigtur

Summary

The protocol calculates the slippage protection value based on the price of OP relative to USD and OP relative to ETH, while the intended exchange is for alUSD and alETH. This results in inaccuracies in the calculated slippage protection value.

Vulnerability Detail

In the `RewardRouter.distributeRewards()` function, the protocol first sends the OP rewards to the OptimismRewardCollector contract,

```
TokenUtils.safeTransfer(IRewardCollector(rewards[vault].rewardCollectorAddress).⌟
↪  rewardToken(), rewards[vault].rewardCollectorAddress, amountToSend);
   rewards[vault].lastRewardBlock = block.number;
   rewards[vault].rewardPaid += amountToSend;
```

then calls the `RewardCollector.claimAndDonateRewards()` function to convert OP into alUSD or alETH.

```
return IRewardCollector(rewards[vault].rewardCollectorAddress).claimAndDonateRew⌟
↪  ards(vault,
↪  IRewardCollector(rewards[vault].rewardCollectorAddress).getExpectedExchange(⌟
↪  vault) * slippageBPS / BPS);
```

During the conversion process, there is a parameter for slippage protection, which is calculated using `OptimismRewardCollector.getExpectedExchange() * slippageBPS / BPS`. Let's take a look at the `getExpectedExchange()` function. In this function, the protocol retrieves the prices of optoUSD and optoETH from Chainlink.

```
(
    uint80 roundID,
    int256 opToUsd,
    ,
    uint256 updateTime,
    uint80 answeredInRound
) = IChainlinkOracle(opToUsdOracle).latestRoundData();
```

```
// Ensure that round is complete, otherwise price is stale.
    (
        uint80 roundIDEth,
        int256 ethToUsd,
        ,
        uint256 updateTimeEth,
        uint80 answeredInRoundEth
    ) = IChainlinkOracle(ethToUsdOracle).latestRoundData();
```

If `debtToken == alUsdOptimism`, the expectedExchange for slippage protection is calculated as totalToSwap * uint(opToUsd) / 1e8.

```
// Find expected amount out before calling harvest
   if (debtToken == alUsdOptimism) {
       expectedExchange = totalToSwap * uint(opToUsd) / 1e8;
```

If debtToken == alEthOptimism, the expectedExchange for slippage protection is calculated as totalToSwap * uint(uint(opToUsd)) / uint(ethToUsd).

```
else if (debtToken == alEthOptimism) {
          expectedExchange = totalToSwap * uint(uint(opToUsd)) / uint(ethToUsd);
```

Here, we observe that the `expectedExchange` is calculated based on the value of OP relative to USD and OP relative to ETH, while the protocol intends to exchange for alUSD and alETH.

```
if (debtToken == 0xCB8FA9a76b8e203D8C3797bF438d8FB81Ea3326A) {
           // Velodrome Swap Routes: OP -> USDC -> alUSD
           IVelodromeSwapRouter.route[] memory routes = new
↪   IVelodromeSwapRouter.route[](2);
           routes[0] =
↪   IVelodromeSwapRouter.route(0x4200000000000000000000000000000000000042,
↪   0x7F5c764cBc14f9669B88837ca1490cCa17c31607, false);
           routes[1] =
↪   IVelodromeSwapRouter.route(0x7F5c764cBc14f9669B88837ca1490cCa17c31607,
↪   0xCB8FA9a76b8e203D8C3797bF438d8FB81Ea3326A, true);
```

```
            TokenUtils.safeApprove(rewardToken, swapRouter, amountRewardToken);

↪   IVelodromeSwapRouter(swapRouter).swapExactTokensForTokens(amountRewardToken,
↪   minimumAmountOut, routes, address(this), block.timestamp);
        } else if (debtToken == 0x3E29D3A9316dAB217754d13b28646B76607c5f04) {
            // Velodrome Swap Routes: OP -> alETH
            IVelodromeSwapRouter.route[] memory routes = new
↪   IVelodromeSwapRouter.route[](1);
            routes[0] =
↪   IVelodromeSwapRouter.route(0x4200000000000000000000000000000000000042,
↪   0x3E29D3A9316dAB217754d13b28646B76607c5f04, false);
            TokenUtils.safeApprove(rewardToken, swapRouter, amountRewardToken);

↪   IVelodromeSwapRouter(swapRouter).swapExactTokensForTokens(amountRewardToken,
↪   minimumAmountOut, routes, address(this), block.timestamp);
        }
```

However, the price of alUSD is not equivalent to USD, and the price of alETH is not equivalent to ETH. This discrepancy leads to inaccuracies in the calculated value for slippage protection, making the protocol vulnerable to sandwich attacks.

Impact

The protocol is susceptible to sandwich attacks.

Code Snippet

https://github.com/sherlock-audit/2024-04-alchemix/blob/main/v2-foundry/src/utils/collectors/OptimismRewardCollector.sol#L120-L126

Tool used

Manual Review

Recommendation

Calculate using the correct prices.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/alchemix-finance/v2-foundry/commit/f0e7530cde2c006fd13c7c34b695113679a9655b

sherlock-admin2

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.