MOVELABS

Meshswap
Security Analysis Report

rev 1.0

Prepared for
Ozys

Prepared by
MOVE LABS

INTRODUCTION

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another party. Any subsequent publication of this report shall be without mandatory consent.

Name	Ozys
Website	https://meshswap.fi/
Repository	https://github.com/meshswap-fi/audit-movelabs.git
Commit	59c75c15dc944bcee8b09cf422fc5a68310819c8
Platform	Polygon
Network	Mainnet
Languages	Solidity
Method	Source code auditing, Automated static analysis
Approver	Aiden Hyun
Timeline	2023-02-13 ~ 2023-04-12
Changelog*	None
Changelog*	None

TABLE OF CONTENTS

Project Overview **4**

About Project 4

Period 4

Project Targets 4

Revision History 4

SCOPE **5**

CHECKLIST **6**

Basic bugs 6

Business logic 6

Specific scenario 6

FINDINGS **7**

Impact classification 7

Summary 7

Issue 1 - *Unexpected behavior can happen during pool initialization* 8

RECOMMENDATIONS **9**

Summary 9

Recommendation 1 - *Remove dead code* 10-11

Recommendation 2 - *Unnecessary require statement* 12

SPECIFIC SCENARIO ANALYSIS **13**

Scenario 1 - *Whether attacker can use custom pool to get mesh reward* 13-14

Scenario 2 - *Sandwich attack while opening position at PlusPool* 15

APPENDIX **16**

Slither 16

Project Overview

About Project

In this project, we conducted a security audit for MeshSwap to improve its security and prevent potential security problems. Our main focus was on verifying whether unauthorized users could access smart contracts that require permission or adversely affect the service. Meshswap is an AMM-based decentralized exchange protocol that was launched in 2022. It is a lending and borrowing protocol that can interact with other EVM-based DeFi DApps. Meshswap provides Swap, Lend, Leverage farm, and Stake on the Polygon network. Users can stake and farm on Meshswap for greater rewards.

Period

- Overall Period
    - 2023-02-13 ~ 2023-04-12

Project Targets

- meshswap
    - Repository : https://github.com/meshswap-fi/audit-movelabs.git
    - Type : Solidity
    - Platforms : Polygon Network

Revision History

Version	Date	Comment
1.0	2023-04-12	Meshswap Audit Report V1

SCOPE

We were provided with a stable source code tree to review. We also reviewed each of the committed fixes.

Source code:

- Meshswap

  o commit SHA
  - 59c75c15dc944bcee8b09cf422fc5a68310819c8

CHECKLIST

We reviewed source code based on the checklist below.

Basic bugs

- Access Control & Authorization
- Ownership Takeover
- Reentrancy
- Integer Overflow/Underflow
- Wrong timestamp implementation
- DoS caused by wrong revert, infinite loop, etc..
- ERC20 idiosyncrasy

Business logic

- Properly implemented functionality
- Incorrect token/fee calculation
- Rounding errors
- Wrong implementation of feature
- Code asymmetries
- synchronized state variables
- Governance token implementation

Specific scenario

- Whether attacker can use custom pool to get mesh reward
- Sandwich attack while opening position at PlusPool

FINDINGS

Impact classification

Severity	Description
High	This vulnerability affects a large number of users and has a critical impact on financial services.
Medium	This vulnerability affects the functionality of financial service but does not result in direct loss of funds.
Low	This vulnerability does not directly affect the service but if combined with other vulnerability can result in severe issue.

Summary

#	Title	Severity
1	Issue 1 - *Unexpected behavior can happen during pool initialization*	Low

Issue 1 - *Unexpected behavior can happen during pool initialization*

Summary	Severity
In ConcentratedFactory.impl.sol, initPool is called after transferFrom is called. If createPool is called with a token that implements transferFrom hook, unexpected behavior can happen.	Low

EIP-777 provides a hook when calling transferFrom function which has been exploited many times as an reentrancy attack. We noticed that during the pool creation, if the attacker creates a pool with EIP-777 token reentrancy can happen. As pool is uninitialized when transferFrom is called, unexpected behavior can occur.

```
address exc = address(new ConcentratedExchange(c.token0, c.token1, c.fee));

IERC20 token0 = IERC20(c.token0);
IERC20 token1 = IERC20(c.token1);

uint bal0 = token0.balanceOf(address(this));
uint bal1 = token1.balanceOf(address(this));
token0.transferFrom(msg.sender, address(this), c.amount0);
token1.transferFrom(msg.sender, address(this), c.amount1);
token0.approve(exc, uint(-1));
token1.approve(exc, uint(-1));

// omitted

IConcentratedExchange(exc).initPool(msg.sender, c.minRatioX96, c.maxRatioX96, c.amount0, c.amount1);
```

File : *ConcentratedFatory.impl.sol* **Function** : *createPool*

Severity is low as there is no clear way an attacker can leverage this issue.

Fix

Call initPool function before calling transferFrom function to prevent unexpected behavior.

RECOMMENDATIONS

These are suggestions to improve code maintainability, readability, and/or resilience.

Summary

#	Title
1	Recommendation 1 - *Remove dead code*
2	Recommendation 2 - *Unnecessary require statement*

Recommendation 1 - *Remove dead code*

We used slither* to perform automated analysis of the codebase. From the result, we found out that there were several dead codes that were not used through the codebase. Following is the result from the tool.

```
## dead-code

- [ ] ID-191
[Address.sendValue(address,uint256)](contracts/BuybackFund.impl.sol#L75-L80) is never
used and should be removed
contracts/BuybackFund.impl.sol#L75-L80

- [ ] ID-196
[Address.toPayable(address)](contracts/BuybackFund.impl.sol#L71-L73) is never used and
should be removed
contracts/BuybackFund.impl.sol#L71-L73

- [ ] ID-211
[FixedPoint.sqrt(FixedPoint.uq112x112)](contracts/MESHswapOracle.sol#L141-L143) is never
used and should be removed
contracts/MESHswapOracle.sol#L141-L143

- [ ] ID-212
[FixedPoint.decode(FixedPoint.uq112x112)](contracts/MESHswapOracle.sol#L125-L127) is
never used and should be removed
contracts/MESHswapOracle.sol#L125-L127

- [ ] ID-213
[FixedPoint.div(FixedPoint.uq112x112,uint112)](contracts/MESHswapOracle.sol#L104-L107) is
never used and should be removed
contracts/MESHswapOracle.sol#L104-L107

- [ ] ID-215
[FixedPoint.encode144(uint144)](contracts/MESHswapOracle.sol#L99-L101) is never used and
should be removed
contracts/MESHswapOracle.sol#L99-L101

- [ ] ID-216
[FixedPoint.encode(uint112)](contracts/MESHswapOracle.sol#L94-L96) is never used and
should be removed
contracts/MESHswapOracle.sol#L94-L96

- [ ] ID-218
[FixedPoint.fraction(uint112,uint112)](contracts/MESHswapOracle.sol#L119-L122) is never
used and should be removed
contracts/MESHswapOracle.sol#L119-L122

- [ ] ID-219
[Context._msgData()](contracts/MESHswapOracle.sol#L153-L156) is never used and should be
removed
```

contracts/MESHswapOracle.sol#L153-L156

- [ ] ID-220
[FixedPoint.**reciprocal**(FixedPoint.uq112x112)](contracts/MESHswapOracle.sol#L135-L138) is never used and should be removed
contracts/MESHswapOracle.sol#L135-L138

- [ ] ID-223
[Babylonian.**sqrt**(uint256)](contracts/MESHswapOracle.sol#L62-L73) is never used and should be removed
contracts/MESHswapOracle.sol#L62-L73

- [ ] ID-224
[Address.**sendValue**(address,uint256)](contracts/MESHswapRouter.impl.sol#L79-L85) is never used and should be removed
contracts/MESHswapRouter.impl.sol#L79-L85

- [ ] ID-229
[EIP2771Recipient.**_msgData**()](contracts/EIP2771Recipient.sol#L35-L48) is never used and should be removed
contracts/EIP2771Recipient.sol#L35-L48

- [ ] ID-230
[Address.**toPayable**(address)](contracts/MESHswapRouter.impl.sol#L74-L76) is never used and should be removed
contracts/MESHswapRouter.impl.sol#L74-L76

- [ ] ID-233
[MESHswapLibrary.**sortTokens**(address,address,address)](contracts/MESHswapRouter.impl.sol#L135-L140) is never used and should be removed
contracts/MESHswapRouter.impl.sol#L135-L140

- [ ] ID-235
[MESHswapLibrary.**getReserves**(address,address,address)](contracts/MESHswapRouter.impl.sol#L146-L150) is never used and should be removed
contracts/MESHswapRouter.impl.sol#L146-L150

Recommendation 2 - *Unnecessary require statement*

During the source code review, we found that the giveAllToUser function has unnecessary require statement which checks whether s.bPrincipal is equal to zero. This is because the check is already done at the higher callstack.

```
502     function closePosition(uint amount, uint p2b, uint c2b, uint p2c) public onlyLeveragedSinglePool positionExists ReentrancyGuard {
503         require(s.pos != Position.UNKNOWN);
504         RepayStatus repayStatus;
505         uint bBefore;
506
507         giveReward();
508
509         bBefore = getBorrowAmountCurrent();
510         repayStatus = repayAll(amount, p2b, c2b);
511
512         if (s.bPrincipal == 0) {
513             giveAllToUser(p2c);
514         }
515
516         emit ClosePosition(s.pos, bBefore.sub(getBorrowAmountCurrent()));
```

File : *LeveragedUser.impl.sol #512* **Function** : *closePosition*

```
429     function giveAllToUser(uint p2c) private {
430         require(s.bPrincipal == 0);
431         uint diff;
432         uint cReturn;
433
434         cReturn = withdrawSinglePoolByAmount(s.cToken, s.icAmount);
435         s.icAmount = 0;
436
437         if (s.ipAmount > 0) {
438             diff = withdrawSinglePoolByAmount(s.pToken, s.ipAmount);
439             s.ipAmount = 0;
440
441             cReturn = cReturn.add((s.pToken != s.cToken) ? exchangePos(s.pToken, diff, s.cToken, p2c) : diff);
442         }
```

File : *LeveragedUser.impl.sol #430* ***Function*** : *giveAllToUser*

SPECIFIC SCENARIO ANALYSIS

Here we will describe how we made our vulnerability assumptions and verified them.

Scenario 1 - *Whether attacker can use custom pool to get mesh reward*

In Exchange.impl.sol, giveReward function is responsible for calculating the reward that should be given to the users who added liquidity to the pool. That is done by getting the balance of lpToken and multiplying it by currentIndex that has been accumulated.

```
function giveReward(address user) private {
    ITreasury(getTreasury()).claim(user, address(this));

    uint lastIndex = userLastIndex[user];
    uint currentIndex = updateMiningIndex();

    uint have = balanceOf[user];

    if (currentIndex > lastIndex) {
        userLastIndex[user] = currentIndex;

        if (have != 0) {
            uint amount = have.mul(currentIndex.sub(lastIndex)).div(10 ** 18);
            IMESH(mesh).sendReward(user, amount);

            userRewardSum[user] = userRewardSum[user].add(amount);
            emit GiveReward(user, amount, currentIndex, userRewardSum[user]);
        }
    }
}
```

File : *Exchange.impl.sol #290-309* **Function** : *giveReward*

We made an assumption that if an attacker could manipulate the balance of lpToken somehow, then it would result in a loss of the mesh fund. To do that, we thought of creating our own custom pool to increase the amount of lpToken. However after creating the pool and calling the giveReward function, we noticed that currentIndex was not updated even if several blocks were mined. By analyzing the source code we found out that currentIndex was calculated with the boostingPower variable which is being set from the getPoolBoosting function.

```
function getPoolBoosting(address pool) public view returns (uint boostingPower) {
    require(pool > address(2));

    address token0 = IExchange(pool).token0();
    address token1 = IExchange(pool).token1();

    address mesh = IGovernance(governance).mesh();
    if((isBoostingToken[token0] && token1 == mesh) || (token0 == mesh && isBoostingToken[token1])){ // MESH - Grade A
        boostingPower = boostingPowerMESH_A;
    }else if(isBoostingToken[token0] && isBoostingToken[token1]){ // Grade A - Grade A
        boostingPower = boostingPowerA_A;
    }else if((token0 == mesh && isValidToken[token1]) || (isValidToken[token0] && token1 == mesh)){ // MESH - Grade B
        boostingPower = boostingPowerMESH_B;
    }else if((isBoostingToken[token0] && isValidToken[token1]) || (isValidToken[token0] && isBoostingToken[token1])){ // Gr
        boostingPower = boostingPowerA_B;
    }else{ // etc ( not mining pool )
        boostingPower = 0;
    }
}
```

File : *PoolVoting.impl.sol #207-225* **Function** : *getPoolBoosting*

Here we can see that boostingPower is not equal to zero only if the token that we used to create a pool is one of boosting tokens, mesh token or valid tokens set by governance or pool operator. Since we couldn't find a way to bypass this check we concluded that it is impossible to get the reward from a custom pool.

Scenario 2 - *Sandwich attack while opening position at PlusPool*

In PlusPool, to open a position, we have to call either openPositionETH or openPositionToken and these two functions call openPositionInternal to handle the logic. Inside the openPositionInternal, approximately 5 things happen.

1. Checks if given borrow amount is available
2. Borrows the token
3. Add liquidity to the pool with the collateral and borrowed token.
4. Swaps the remaining token and adds them to the pool again.
5. Updates the userLP balance and checks debt ratio.

Our assumption here was that if we borrow the token as much as possible and maximize the remaining token for swapping, then we can abuse the AMM formula X * Y = K to perform the sandwich attack. However after testing the scenario we noticed that this doesn't work because a debt ratio check is done at last to see whether the price of lpToken is within the liquidation price.

APPENDIX

Slither

We used Slither as an automated analysis tool, and below is the test environment and tool usage.

- **Repository** : https://github.com/crytic/slither.git

- **Test Environment** : Linux ubuntu-20 5.15.0-67-generic
  - How to install

```
sudo apt install python3-pip
python3 -m pip install slither-analyzer
pip3 install solc-select
solc-select install 0.5.6
solc-select use 0.5.6
```

- **Usage**
  - How to run

```
slither ./contracts/ --checklist >> ./results.txt
cat results.txt
```

- **Example**

```
 - [ ] ID-1
[Factory.constructor(address,address,address,address)._implementation](contracts/Factory.sol#L32) lacks a zero-check on :
                - [implementation = _implementation](contracts/Factory.sol#L38)

contracts/Factory.sol#L32


 - [ ] ID-2
[Factory.constructor(address,address,address,address)._mesh](contracts/Factory.sol#L34) lacks a zero-check on :
                - [mesh = _mesh](contracts/Factory.sol#L39)

contracts/Factory.sol#L34


 - [ ] ID-3
[Factory.constructor(address,address,address,address)._WETH](contracts/Factory.sol#L35) lacks a zero-check on :
                - [WETH = _WETH](contracts/Factory.sol#L42)

contracts/Factory.sol#L35
```