Protocol Audit Report

Version 1.0

Cyfrin.io

January 31, 2024

T-Swap Audit Report

0xShitgem

January 31, 2024

Lead Auditors:

- 0xShitgem

Table of Contents

Protocol Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

TSwap Pools

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

Disclaimer

The 0xShitgem makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by me is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

I'm using the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.sol
```

Roles

- Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made.
- Users: Users who want to swap tokens.

Executive Summary

Issues found

Findings

High

[H-01] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees

Description: The `getInputAmountBasedOnOutput` function is calculating what input needs to be passed base on outputAmount. There is however incorrect calulation in returning value. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: Any function using this function will have more fees then expected from users.

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
    uint256 outputAmount,
    uint256 inputReserves,
    uint256 outputReserves
)
    public
    pure
    revertIfZero(outputAmount)
    revertIfZero(outputReserves)
    returns (uint256 inputAmount)
{
-   return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997);
+   return ((inputReserves * outputAmount) * 1000) / ((outputReserves - outputAmount) * 997);
}
```

[H-02] `TSwapPool::swapExactOutput` doesn't have slippage correction, resulting in potentially less tokens

Description:

Found in src/TSwapPool.sol [Line 348]

```
function swapExactOutput(
    IERC20 inputToken,
    IERC20 outputToken,
    uint256 outputAmount,
    uint64 deadline
)
    public
    revertIfZero(outputAmount)
    revertIfDeadlinePassed(deadline)
    returns (uint256 inputAmount)
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

    inputAmount = getInputAmountBasedOnOutput(
        outputAmount,
        inputReserves,
        outputReserves
    );

    _swap(inputToken, inputAmount, outputToken, outputAmount);
}
```

`swapExactOutput` function “figures out how much you need to input based on how much output you want to receive”. The thing is function doesn’t provide security checks on amount you provide - so-called slippage protection. User could pass any amount of tokens he wants to.

Impact: Attacker by passing absurdly high amount of tokens can fluctuate prices which is base of “sandwich attacks”. Impact for user is that market conditions can drastically change resulting in much less tokens.

Proof of Concept: There’s sandwich attack which happen in 1 block:

1. Attacker manipulates pool price
2. User buy at bad price
3. Attacker sell at higher price
4. User is left with much less tokens then expected

Recommended Mitigation: Add slippage control checks

```
+error TSwapPool__SlippageProtection(
+   IERC20 inputToken,
+   IERC20 outputToken,
+   uint256 maxInputAmount,
+);

function swapExactOutput(
    IERC20 inputToken,
    IERC20 outputToken,
+   uint256 maxInputAmount,
    uint256 outputAmount,
    uint64 deadline
)
    public
    revertIfZero(outputAmount)
    revertIfDeadlinePassed(deadline)
    returns (uint256 inputAmount)
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

    inputAmount = getInputAmountBasedOnOutput(
        outputAmount,
        inputReserves,
        outputReserves
    );

+   if (inputAmount > minInputAmount){
+       revert TSwapPool__SlippageProtection(inputToken, outputToken, maxInputAmount)
+   }

    _swap(inputToken, inputAmount, outputToken, outputAmount);
}
```

[H-03] `TSwapPool::swapExactOutput` parameters inside `TSwapPool::sellPooltokens` are passed backwards resulting in wrong return value

Description:

On [Line 378] there is below function.

```
function sellPoolTokens(
    uint256 poolTokenAmount
) external returns (uint256 wethAmount) {
    return
        swapExactOutput(
            i_poolToken,
            i_wethToken,
            poolTokenAmount,
            uint64(block.timestamp)
        );
    }
```

However the `swapExactOutput` function miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` function is the one that should be called, because users specify the exact amount of input tokens, not output.

Impact: Users will swap the wrong amount of token.

Recommended Mitigation: Change `swapExactOutput` to `swapExactInput`.

```
function sellPoolTokens(
    uint256 poolTokenAmount
+   uint256 minWethToReceive
) external returns (uint256 wethAmount) {
    return
-       swapExactOutput(
-           i_poolToken,
-           i_wethToken,
-           poolTokenAmount,
-           uint64(block.timestamp)
-       );
+       swapExactInput(
+           i_poolToken,
+           poolTokenAmount,
+           i_weth,
+           minWethToReceive,
+           uint64(block.timestamp)
+       );
}
```

[H-04] In `TSwapPool::_swap` the extra token given to users after every `swapCount` break protocol invariant of `x * y = k`

Description: The protocol follow a strict invariant of `x * y = k`. Where:

- `x`: The balance of the pool token
- `y`: The balance of WETH
- `k`: The constant product of the two balances.

This invariant should never be broken, however inside `_swap` function as documention says "Every 10 swaps, we give the caller an extra token as an extra incentive to keep trading on T-Swap."

From more developer persepctive there exist `swapCount` which increment each times user makes swap. After 10 swap token is send to `msg.sender`. This type of incentive breaks completely invariant

Impact A user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given out by the protcol.

Proof of Concept:

1. User makes 10 swaps
2. Users get incentive for trading on TSwap
3. Invariance is broken - User continues to swap until all the protocol funds are drained.

Proof Of Code

```
function testInvariantBroken() public {
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), 100e18);
    poolToken.approve(address(pool), 100e18);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
    vm.stopPrank();

    uint256 outputWeth = 1e17;

    vm.startPrank(user);
    poolToken.approve(address(pool), type(uint256).max);
    poolToken.mint(user, 100e18);
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));

    int256 startingY = int256(weth.balanceOf(address(pool)));
    int256 expectedDeltaY = int256(-1) * int256(outputWeth);

    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    vm.stopPrank();

    uint256 endingY = weth.balanceOf(address(pool));
    int256 actualDeltaY = int256(endingY) - int256(startingY);
    assertEq(actualDeltaY, expectedDeltaY);
}
```

Recommended Mitigation: Remove the extra incentive. If you want to keep this in, we should account for the change in the `x * y = k` protocol invariant. Or, we should set aside tokens in the same way we do with fees.

Medium

[M-01] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline

Description:

Found in src/TSwapPool.sol [Line 117]

```
function deposit(
    uint256 wethToDeposit,
    uint256 minimumLiquidityTokensToMint,
    uint256 maximumPoolTokensToDeposit,
    uint64 deadline
) {}
```

`deadline` inside `TSwapPool::deposit` is passed as argument and provided natspec for it, but never used inside function. According to documentation: “The deadline for the transaction to be completed by”. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable to deposit, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is unused.

Recommended Mitigation:

```
function deposit(
    uint256 wethToDeposit,
    uint256 minimumLiquidityTokensToMint,
    uint256 maximumPoolTokensToDeposit,
    uint64 deadline
)
    external
+   revertIfDeadlinePassed(deadline)
    revertIfZero(wethToDeposit)
    returns (uint256 liquidityTokensToMint)
{
}
```

[M-02] Rebase, fee-on-transfer and ERC777 tokens break protocol invariance

Description: The protocol invariance is describe as “Constant Product Formula” a.k.a `x*y=k`. However there exist some tokens that break that protocol invariance and they’re:

- Rebasing tokens - everytime price goes up - it mints more token and if price goes down - it burn more tokens,
- Fee-on-transfer tokens - as name suggest, everytime we make transfer with token it takes some fee
- ERC777 - these are tokens that enable reentrency

Impact: Usage of these tokens in TSwap would be potential to drain entire liquidity pools by attacker.

Recommended Mitigation:

1. Consider blocking these tokens from interacting with protocol.
2. Consider making checks for every of these tokens

    - For rebasing and fee-on-transfer tokens compare pre/after balances to compute the actual deposited amount
    - For ERC777 add reentrancy guards

Low

[L-01] `TSwapPool::LiquidityAdded` event's parameters are in wrong place

Description: Found in src/TSwapPool.sol [Line 203]

```
emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
```

When emitting `LiquidityAdded` in the `TSwapPool::_addLiquidityMintAndTransfer` function, it logs values in an incorrect order.

The correct way of this event is:

```
event LiquidityAdded(
    address indexed liquidityProvider,
    uint256 wethDeposited,
    uint256 poolTokensDeposited
);
```

Impact: Event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation:

```
-   emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+   emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);
```

[L-04] `TSwapPool::poolTokenReservers` isn't used in function, making contract use more gas

Description:

Found in src/TSwapPool.sol [Line 138]

```
uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```

`poolTokenReserves` isn't used anywhere inside function.

Impact: It waste gas

Recommended Mitigation: Consider deleting `poolTokenReservers`

[L-03] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given

Description: The function `swapExactInput` is expected to return `uint256 output` which is actual amount of tokens bought by the caller. However in function there's no return statement.

Impact: The function will always return 0, giving incorrect information to the caller.

Recommended Mitigation:

```
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

-   uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);
+   output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);

-   if (output < minOutputAmount) {
-       revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
+   if (output < minOutputAmount) {
+       revert TSwapPool__OutputTooLow(output2, minOutputAmount);
    }

-   _swap(inputToken, inputAmount, outputToken, outputAmount);
+   _swap(inputToken, inputAmount, outputToken, output);
    }
```

Informational

[I-01] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used

Found in src/PoolFactory.sol [Line: 24]

```
error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

Recommended Mitigation

```
- error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-02] Lacking zero address checks

Found in src/PoolFactory.sol [Line: 43]

```
constructor(address wethToken) {
    i_wethToken = wethToken;
}
```

Found in src/TSwapPool.sol [Line 92]

```
constructor(
    address poolToken,
    address wethToken,
    string memory liquidityTokenName,
    string memory liquidityTokenSymbol
) ERC20(liquidityTokenName, liquidityTokenSymbol) {
    i_wethToken = IERC20(wethToken);
    i_poolToken = IERC20(poolToken);
}
```

Recommended Mitigation

```
constructor(address wethToken) {
+   if(wethToken == address(0)) {
+       revert();
+   }
    i_wethToken = wethToken;
}
```

[I-03] Unnecessary constant variable passed to `TSwapPool::TSwapPool__WethDepositAmountTooLow` event

Found in src/TSwapPool.sol [Line 130]

```
revert TSwapPool__WethDepositAmountTooLow(
    MINIMUM_WETH_LIQUIDITY,
    wethToDeposit
);
```

`TSwapPool::MINIMUM_WETH_LIQUIDITY` is constant, so it's unnecessary to passing it inside event parameters.

[I-04] `TSwapPool::deposit` doesn't follow CEI

Description: Found in src/TSwapPool.sol [Line 187]

```
_addLiquidityMintAndTransfer(
    wethToDeposit,
    maximumPoolTokensToDeposit,
    wethToDeposit
);

liquidityTokensToMint = wethToDeposit;
```

Impact Despite that not following here CEI doesn't have consequences - better is to follow that pattern.

[I-05] `createPool` should use `.symbol()` instead of `.name()`

Description:

Found in src/PoolFactory.sol [Line 57]

```
string memory liquidityTokenSymbol = string.concat("ts", IERC20(
    tokenAddress).name());
```

Recommended Mitigation

```
-   string memory liquidityTokenSymbol = string.concat("ts", IERC20(
    tokenAddress).name());
+   string memory liquidityTokenSymbol = string.concat("ts", IERC20(
    tokenAddress).symbol());
```

[I-06] Using "magic" numbers instead of variables with clear names

Found in src/TSwapPool.sol [Line 264]

```
uint256 inputAmountMinusFee = inputAmount * 997;
uint256 numerator = inputAmountMinusFee * outputReserves;
uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
```

Better is to use variables with clear names instead of only numbers.

Recommended Mitigation Use variables with clear names

```
+   uint256 FEE_AMOUNT = 997;
+   uint256 FULL_AMOUNT = 1000;

-   uint256 inputAmountMinusFee = inputAmount * 997;
+   uint256 inputAmountMinusFee = inputAmount * FEE_AMOUNT;
uint256 numerator = inputAmountMinusFee * outputReserves;
-   uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
+   uint256 denominator = (inputReserves * FULL_AMOUNT) +
    inputAmountMinusFee;
```

[I-07] Function without natspec

Found in src/TSwapPool.sol [Line 308]

```
function swapExactInput(
    IERC20 inputToken,
    uint256 inputAmount,
    IERC20 outputToken,
    uint256 minOutputAmount,
    uint64 deadline
) {}
```

Gas