T-Swap Initial Audit Report

Version 0.1

jasonschwarz.xyz

June 23, 2024

T-Swap Audit Report

Jason Schwarz

June 23, 2024

T-Swap Audit Report

Prepared by: Jason Schwarz Lead Auditors:

- Jason Schwarz

Assisting Auditors:

- None

Table of contents

About Jason Schwarz

I am a blockchain security researcher and auditor with a focus on smart contract security. I have a background as a Solidity developer and have been working in the blockchain space for over 4 years. I have conducted numerous security audits for various projects and have a deep understanding of the security risks associated with smart contracts.

Disclaimer

I make all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the the findings provided in this document. A security audit 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```
d1783a0ae66f4f43f47cb045e51eca822cd059be
```

Scope

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

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.

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

Severity	Number of issues found
High	5
Medium	0
Low	3
Info	8
Gas Optimizations	1
Total	17

Findings

High

[H-1] Deadline Check Absent in `TSwapPool::deposit`, Allowing Transactions After Deadline

Description: Although the `deposit` function includes a deadline parameter, specified as the transaction's completion time, it is not implemented in the function. This oversight allows liquidity additions to happen at unintended times, risking execution under adverse market conditions.

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

Tools Used: Manual Review

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

Recommended Mitigation: Consider making the following change to the function.

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

[H-2] Incorrect Fee Scaling in TSwapPool::getInputAmountBasedOnOutput Causes Excessive Token Withdrawals

Description: The `getInputAmountBasedOnOutput` function is designed to determine the number of input tokens a user must provide to receive a specified amount of output tokens. However, a critical error in the fee calculation miscalculates the required input. The function mistakenly scales the fee by 10,000 instead of 1,000 basis points, leading to excessive token deductions and user overcharges.

Impact: Protocol takes more fees than 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-3] `TSwapPool::sellPoolTokens` Miscalculates Tokens Due to Incorrect Function Call.

Description: The sellPoolTokens function currently calls the swapExactOutput function, which is incorrect because it passes the poolTokenAmount as the parameter. Since poolTokenAmount specifies the amount of input tokens (pool tokens) the user wants to sell, the function should instead call swapExactInput. This mismatch leads to an incorrect calculation of the token amount received by the user, resulting in users getting incorrect token amounts.

Impact: Users will receive an incorrect amount of tokens due to the miscalculation, severely disrupting protocol functionality.

Recommended Mitigation:

Consider implementing swapExactInput in place of swapExactOutput. This change requires modifying the sellPoolTokens function to include a new parameter, such as minWethToReceive, which will be passed to swapExactInput.

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

[H-4] `TSwapPool::_swap` provides extra tokens each time the `swapCount` condition is met, violating the constant product invariant

Description: The protocol adheres to the invariant x * y = k, where: x: The pool's token balance y: The pool's WETH balance k: The constant product of these two balances

Whenever the pool balances change, the ratio between the token and WETH amounts must remain constant to preserve the invariant x * y = k. However, the _swap function disrupts this balance by providing extra tokens as incentives each time the swapCount condition is met. This breaks the invariant over time, leading to a potential drain of the protocol's funds.

The following code block is causing the issue:

```
        swap_count++;
        if (swap_count >= SWAP_COUNT_MAX) {
            swap_count = 0;
            outputToken.safeTransfer(msg.sender, 1
                _000_000_000_000_000_000);
```

```
5           }
```

Impact: Users can exploit the protocol by performing numerous swaps to repeatedly collect the additional incentives, potentially draining the protocol's funds.

Proof of Concept:

1. A user performs 10 swaps and accumulates an additional 1_000_000_000_000_000_000 tokens as an incentive.
2. The user continues to swap repeatedly until the protocol's funds are completely drained.

Proof Of Code

Place the following into `TSwapPool.t.sol`.

```
1
2       function testInvariantBroken() public {
3           vm.startPrank(liquidityProvider);
4           weth.approve(address(pool), 100e18);
5           poolToken.approve(address(pool), 100e18);
6           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7           vm.stopPrank();
8
9           uint256 outputWeth = 1e17;
10
11          vm.startPrank(user);
12          poolToken.approve(address(pool), type(uint256).max);
13          poolToken.mint(user, 100e18);
14          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
15          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
16          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
17          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
18          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
19          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
20          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
21          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
22          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
23
24          int256 startingY = int256(weth.balanceOf(address(pool)));
25          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
26
```

```
27          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
28          vm.stopPrank();
29
30          uint256 endingY = weth.balanceOf(address(pool));
31          int256 actualDeltaY = int256(endingY) - int256(startingY);
32          assertEq(actualDeltaY, expectedDeltaY);
33      }
```

Recommended Mitigation: Remove the extra incentive mechanism. If retaining it is essential, adjust the protocol to maintain the x * y = k invariant. Alternatively, allocate a separate pool of tokens for incentives, similar to how fees are managed.

```
1 -         swap_count++;
2 -         // Fee-on-transfer
3 -         if (swap_count >= SWAP_COUNT_MAX) {
4 -             swap_count = 0;
5 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
6 -         }
```

[H-5] Lack of Slippage Protection in TSwapPool::swapExactOutput May Result in Users Receiving Significantly Fewer Tokens

Description: The `swapExactOutput` function lacks slippage protection. Unlike the swapExactInput function, which specifies a minOutputAmount, the `swapExactOutput` function should specify a maxInputAmount to safeguard users against unfavorable price movements during transaction processing.

Impact: Users could suffer substantial losses if market conditions shift unfavorably before the transaction completes, resulting in a much worse exchange rate than anticipated.

Proof of Concept:

Assume the current price of 1 WETH is 1,000 USDC. A user initiates a `swapExactOutput` to obtain 1 WETH with the following parameters: inputToken: USDC outputToken: WETH outputAmount: 1 WETH deadline: A future timestamp The function does not specify a maxInputAmount. While the transaction is pending in the mempool, the market conditions change drastically, and the price of 1 WETH increases to 10,000 USDC. The transaction executes, and the user ends up spending 10,000 USDC instead of the anticipated 1,000 USDC.

Recommended Mitigation: Implement a maxInputAmount parameter in the `swapExactOutput` function to limit the amount of tokens the user can spend, allowing them to predict and cap their expenditure.

Low Risk Findings

[L-1] 'TSwapPool::_addLiquidityMintAndTransfer' emits LiquidityAdded event with misMatched parameters

Description: The `LiquidityAdded` event is emitted with mismatched parameters in the `TSwapPool::_addLiquidityMintAndTransfer` function. The `poolTokensToDeposit` and `wethToDeposit` parameters are swapped in the event emission.

Impact: Event emission with mismatched parameters can lead to confusion and misinterpretation of the event data by off-chain consumers.

Recommended Mitigation:

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

[L-2] Missing Return Value from TSwapPool::swapExactInput causes users always recieve default value as output.

Description: The swapExactInput function is intended to return the actual amount of tokens purchased by the caller. However, despite declaring a named return value output, it neither assigns a value to output nor uses an explicit return statement.

Impact: The return value will always be 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(outputAmount, minOutputAmount);
         }
```

```
13
14 -         _swap(inputToken, inputAmount, outputToken, outputAmount);
15 +         _swap(inputToken, inputAmount, outputToken, output);
16       }
```

[L-3] Lack of zero address validation

Impact: Constructors do not validate against zero addresses. Since the constructor parameter initializes a state variable used in multiple functions throughout the contract, this omission can propagate errors, potentially necessitating contract redeployment.

Tools Used: Manual Review

Recomended Migration: Add require condition to validate against zero address

2 Found Instances

- Found in src/TSwapPool.sol Line: 91

```
1         constructor(
2         address poolToken,
3         address wethToken,
4         string memory liquidityTokenName,
5         string memory liquidityTokenSymbol
6     ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
7         i_wethToken = IERC20(wethToken);
8         i_poolToken = IERC20(poolToken);
9     }
```

- Found in src/PoolFactory.sol Line: 41

```
1         constructor(address wethToken) {
2         i_wethToken = wethToken;
3         }
```

```
1     constructor(address wethToken) {
2 +       require(wethToken != address(0), "TSwapPool__ZeroAddress");
3         i_wethToken = wethToken;
4     }
```

Informational

[I-1] `poolTokenReserves` is not used in `TSwapPool::deposit` and should be removed

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

[I-2]: Define and use `constant` variables instead of using literals

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract to reduce the risk of errors and improve readability.

4 Found Instances

- Found in src/TSwapPool.sol Line: 276

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 295

```
            ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 454

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
                1e18,
```

I-3: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

```
    event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1      event Swap(
```

I-4: PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

2 Found Instances

- Found in src/PoolFactory.sol Line: 15

```
1 pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
1 pragma solidity 0.8.20;
```

I-5: `T-Swap::PoolFactory` contains unused Custom Error

It is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/PoolFactory.sol Line: 22

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

I-6: Multiple compiler versions allowing a wide range from 0.6.2 to >=0.8.20

Multiple compiler versions are used in the project, which can lead to inconsistencies in the compiled bytecode. It is recommended to use a single compiler version to avoid potential issues.

3 Found Versions

```
1 - Version constraint >=0.6.2 is used by:
2     - lib/forge-std/src/interfaces/IERC20.sol#2
3 - Version constraint ^0.8.20 is used by:
4     - lib/openzeppelin-contracts/contracts/interfaces/draft-IERC6093.
        sol#3
5     - lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#4
6     - lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4
```

```
7       - lib/openzeppelin-contracts/contracts/token/ERC20/extensions/
          IERC20Metadata.sol#4
8       - lib/openzeppelin-contracts/contracts/token/ERC20/extensions/
          IERC20Permit.sol#4
9       - lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.
          sol#4
10      - lib/openzeppelin-contracts/contracts/utils/Address.sol#4
11      - lib/openzeppelin-contracts/contracts/utils/Context.sol#4
12  - Version constraint 0.8.20 is used by:
13      - src/PoolFactory.sol#15
14      - src/TSwapPool.sol#15
```

I-7: Natspec comments are missing throughout the codebase

Natspec comments are used to provide information about the contract, its functions, and its variables. They are used to generate documentation for the contract and are important for developers to understand the contract's functionality.

[I-8] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

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

Gas Optimizations

[G-1] `Use function instead of modifiers (4 instances)`

Description: Modifiers often duplicate bytecode. Each time a modifier is used, its code is inlined into every function it modifies, leading to repetitive bytecode in the final contract. This increases the overall contract size and execution cost. Functions, on the other hand, promote code reuse. When a function is called, the EVM executes a single copy of the function's bytecode, avoiding duplication. This reduces the size of the contract bytecode and the gas used for execution.

Impact: The contract size and execution cost are increased due to the repetitive bytecode generated by modifiers. Gas costs are higher than necessary.

Recommended Mitigation:

```
1  -    modifier revertIfZero(uint256 amount) {
```

```
-         if (amount == 0) {
-             revert TSwapPool__MustBeMoreThanZero();
-         }
-         _;
-     }

+     function revertIfZero(uint256 amount) internal pure {
+         require(amount != 0, "TSwapPool__MustBeMoreThanZero");
+     }

    function withdraw(
        uint256 liquidityTokensToBurn,
        uint256 minWethToWithdraw,
        uint256 minPoolTokensToWithdraw,
        uint64 deadline
    )
        external
        revertIfDeadlinePassed(deadline)
-         revertIfZero(liquidityTokensToBurn)
-         revertIfZero(minWethToWithdraw)
-         revertIfZero(minPoolTokensToWithdraw)
    {
+         revertIfZero(liquidityTokensToBurn);
+         revertIfZero(minWethToWithdraw);
+         revertIfZero(minPoolTokensToWithdraw);
```