T-Swap 監査 レポート

Version 1.0

Jeremy Bru ・ジェレミー　ブルー

February 29, 2024

T-Swap 監査 レポート

Jeremy Bru ・ジェレミー　ブルー

2024 年 2 月 29 日

最小限の監査レポート・T-Swap

作成者: Jeremy Bru (Link)

主任監査役・リードセキュリティ担当:

- Jeremy Bru

Contact: 一

目次

- 問題の発見
  - 高
  - 中
  - 低
  - 情報系
  - ガス

プロトコル概要

私の理解によると、このプロジェクトは、ユーザーが公正な価格で互いに資産を交換できる許可のない方法を意図しています。プロトコルは AMM（Automated Market Maker：自動市場作成者）であり、以下のことを行うべきです：

- 各プールについて、定数式 `x * y = k` を尊重する。
  - トークンの比率は常に同じままであるべきです。
- プロトコルが機能するためには、流動性提供者が必要です。
  - 彼らのシェアは `LP` ERC20 トークンによって表されます。例えば `T-SwapWeth`。
  - スワップが行われるたびに 0.3% の手数料を得ます。
  - 流動性提供者への生成された利益は、流動性プールトークン (`LP`s) 数の増加の形で基づくことになります。
- 流動性は、これら 2 つの関数によって預けられ、追加され、引き出されることができます：
  - `TSwapPool::deposit`
  - `TSwapPool::withdraw`
  - 注：ユーザーからの流動性が存在している場合、預け入れ機能は 流動性の追加 と見なされます。

`PoolFactory` コントラクトは、`PoolFactory::createPool` 関数を介して新しい「プール」のトークンを作成するために使用されるコントラクトです。すべてのプールトークンが正しいロジックを使用していることを確認するのに役立ちます。

- プールはトークン A（x）と WETH トークン（y）の 2 つの資産で構成されています。
- 公正な価格で互いに資産を交換できます。

ユーザーがプール内のトークンを交換するために呼び出すことができる 2 つの関数が `TSwapPool` コントラクトからあります：

- `TSwapPool::swapExactInput`
- `TSwapPool::swapExactOutput`
- ユーザーは、受け取りたいトークンの量または交換したい資産のために提供したいトークンの量に基づいてトークンを交換することができます。

デプロイされるチェーンは `Ethereum` であり、WETH をペアリングトークンとして使用してプールを作成するために `任意のERC20` トークンを使用することができます。

免責事項

私、Jeremy Bru は、与えられた期間内にコードの脆弱性をできるだけ多く見つけるために全力を尽くしましたが、本書類に提供された発見に対しては一切の責任を負いません。チームによるセキュリティ監査は、基盤となるビジネスや製品の推薦ではありません。監査は時間を区切って行われ、コードのレビューはコントラクトの Solidity 実装のセキュリティ側面にのみ焦点を当てて行われました。

リスク分類

		インパクト		
		高	中	低
	高	高	高/中	中
可能性	中	高/中	中	中/低
	低	中	中/低	低

CodeHawks (Link) の重大度マトリックスを使用して重大度を判断します。詳細については、ドキュメンテーションを参照してください。

監査の詳細

Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

スコープ

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

役割

- Swapper - プール内の 2 つの資産間で交換するユーザー。
- 流動性提供者 - プールに流動性を初期化して提供し、ユーザーが資産を交換できるようにするユーザー。彼らのシェアに基づいて、各スワップに手数料を得ます。

エグゼクティブサマリー

Forge Foundry、Aderyn、Slither、状態付きファジングとハンドラー、および手動レビューを使用して、以下の問題を見つけ、問題を示すテストケースを書きました。

発見された問題

深刻度	見つかった問題の数
高	6
中	2
低	3
情報系	10
———	————————
合計	21

問題の発見

- S = Severity: 深刻度
- クリティカル・クリット (Crit)= 非常に高い
- 情報系　= お知らせ事項
- 例：S-低＋番号 = 順番的に並んでいる。

高

[S-高 1] _swap 関数のインセンティブによって核心的不変量 x * y = k が破られ、プールの比率が崩れる

説明:

プロトコルは、swappers（ユーザー）が交換するたびに、WETH トークンで追加のインセンティブを提供しています。これは、ユーザーがプロトコル上で資産を交換し続けるようにするためです。

インセンティブは、10トランザクションごとに1WETHです。

したがって、2 つの資産間で一定の比率を保持するべき不変量x * y = kは、上記のインセンティブによって破られます。

ユーザーは得られるはずのもの以上を得ているため、プールの比率は完全に崩れます。

影響:

- プールは、swappersによって大きな努力なしに 0 に排水される可能性があります。
- そして、これはユーザーごとに 10 回目のトランザクションごとに発生します。

概念実証:

- テストフォルダには、状態付きファジングとハンドラーメソッドを使用して問題を示すテストケースがあります。
- また、ファジングメソッドに基づかない第 3 のテストも追加しました。これは単に 10 回スワップを行うだけです。

最初のテスト:

- test/Invariant.t.sol::statefulFuzz_constantProductFormulaStaysTheSameTokenA

- 最初のペアトークンに基づいて、プールの量の変化と預けられたものの等式をチェックします。
- 等式は一致します。

テストを実行するには:

```
forge test --mt statefulFuzz_constantProductFormulaStaysTheSameTokenA -vvvv
```

2 番目のテストは、Weth トークンに対して同じ検証を目指します。

- test/Invariant.t.sol::statefulFuzz_constantProductFormulaStaysTheSameWeth
- インセンティブのため、等式は一致しません。テストを実行するには:

```
forge test --mt statefulFuzz_constantProductFormulaStaysTheSameWeth -vvvv
```

インセンティブTSwapPool::_swap: 開発者コメント: * @dev 10回のスワップごとに、T-Swapで取引を続けるための追加のインセンティブとして、発信者に追加のトークンを提供します。

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

第 3 のテスト TSwapPool.t.sol:

Proof of Code

```

    function testDepositSwapAudit() public {
        uint256 swapOutput = 1e17;

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

        vm.startPrank(user);
        poolToken.approve(address(pool), 1000e18);
        for (uint256 i = 0; i < 8; i++) {
            console.log("Swap number: ", i);
            pool.swapExactOutput(poolToken, weth, swapOutput, uint64(
                block.timestamp));
        }

        int256 startingWethBalance = int256(weth.balanceOf(address(pool
            )));
        int256 expectedWethBalanceChange = int256(-1) * int256(
            swapOutput);
```

```
20
21          console.log("Swap number: 9");
22          pool.swapExactOutput(poolToken, weth, swapOutput, uint64(block.
                timestamp));
23
24          // comment swap number 10 and the test will pass
25          // the purpose of this test is to show the invariant breaking
                due to the incentives coming in on the 10th swap
26          console.log("Swap number: 10"); // when incentive comes in
27          pool.swapExactOutput(poolToken, weth, swapOutput, uint64(block.
                timestamp));
28
29          vm.stopPrank();
30
31          uint256 endingWethBalance = weth.balanceOf(address(pool));
32          int256 actualWethBalanceChange = int256(endingWethBalance) -
                int256(startingWethBalance);
33
34          assertEq(actualWethBalanceChange, expectedWethBalanceChange);
35      }
```

推奨される軽減策:

- インセンティブロジックを変更するか、削除してください。プールの比率とプロトコルの核心的不変量を破っています。
- ユーザーは、プロトコルのために作られた TSwap トークンで報酬を受け取ることができます。その後、ユーザーが TSwap トークンを Weth トークンに交換できるように、TSwap / Weth プールを作成することができます。
- スワップの最小要件や流動性プロバイダーになるための最小要件との数学的問題を避けるために、インセンティブ額を下げてください。

[S-高 2] スワップインセンティブによるグリーフィング攻撃、ユーザー == スワッパー によってプールが空になり、プロトコルは損失を生き残れません。

説明:

- 流動性提供者が WETH の流動性を預けるために必要な最低限は `1_000_000_000` wei です。
- 10 回のスワップごとに、スワッパーは `1_000_000_000_000_000_000` wei の WETH で報酬を受け取ります。
- 流動性提供者は少なくとも 0.000000001 weth を預ける必要があります <–> スワッパーは 1WETH を得ます

少なくとも 10 回スワップするだけで、スワッパーはプールを 0 にすることができます。

影響:

- スワッパーはどのプールも 0 にすることができます。

概念実証:

`TSwapPool::_swap`からのインセンティブコード：

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

`TSwapPool::deposit`からの最低預金チェッカー：

```
if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
    revert TSwapPool__WethDepositAmountTooLow(
        MINIMUM_WETH_LIQUIDITY,
        wethToDeposit
    );
}
```

`TSwapPool.sol`からの定数 `MINIMUM_WETH_LIQUIDITY`：

```
uint256 private constant MINIMUM_WETH_LIQUIDITY = 1_000_000_000;
```

推奨される軽減策:

- このレポートの最初の問題と同じです、以下をご覧ください：
- インセンティブのロジックを変更するか、削除してください。これはプール比率とプロトコルのコア不変条件を破壊しています。

- ユーザーは TSwap トークンで報酬を受けることができます。これはプロトコルのために作られ、当初はスワッパーによってのみ獲得されミントされるトークンです。その後、TSwap / Weth プールを作成して、ユーザーが TSwap トークンを Weth トークンと交換できるようにすることができます。
- スワップの最小要件や流動性提供者になるための最小要件との数学的問題を避けるために、インセンティブの額を下げてください。

[S-高 3] `TSwapPool::deposit` 関数の未使用パラメータ `deadline`、期限切れ後にトランザクションが通過しないはずというロジックの混乱。

説明:

誰かが将来のブロックにデッドラインを設定した場合、現在のブロックで預金でき、設定したブロックのデッドラインまで預金を続けることができます。ユーザーが何かを期待していても、実際のコードでは常にその通りになります。

- 修飾子`revertIfDeadlinePassed`も`TSwapPool::deposit`関数で使用されていません。
- `uint64 deadline`パラメータは`TSwapPool::deposit`関数で使用されていません。

影響:

- ユーザーがデッドラインのために預金が失敗することを期待している場合、それは起こりません。トランザクションは通過します。ユーザーが何かを期待していても、実際のコードでは常にその通りになります。
- ユーザーはいつでも好きなときに出金できる必要があります。それを説明する権利確定や出金までの最小待機時間はありません。

概念実証:

- 以下は、`user` が `TSwapPool::deposit` 関数を呼び出してプールに流動性を預けるシンプルなテストです。この際、`deadline` はすでに `1000` 秒後の未来に設定されています。
- 預金は成功します。しかし、以下の 推奨される対策 で説明されているように、関数シグネチャにデッドラインをチェックする修飾子が設定されている場合、失敗します。

Proof Of Code

```
    function testBlockTimestamp() public {
        uint256 minimumDeposit = 1e18;
        uint256 currentBalanceOfWethInPool = weth.balanceOf(address(
            pool));

        vm.getBlockTimestamp();
        uint64 currentBlock = uint64(block.timestamp);
        console.log("Current block timestamp: ", currentBlock);

        // increase timestamp
        vm.warp(1000);
        vm.assume(currentBlock != block.timestamp);
        require(currentBlock != block.timestamp);

        // as pool and liquidity already exist (see initial setup
            config)
        // prank with user for just depositing on the initial block.
            timestamp value when deposing at a time where the
        // blockchain is already at a different block forward.
        vm.startPrank(user);
        weth.approve(address(pool), minimumDeposit);
        poolToken.approve(address(pool), minimumDeposit);
        pool.deposit(minimumDeposit, 0, minimumDeposit, currentBlock);
        vm.stopPrank();

        //check deposit by checking that the liquidity pool has
            increased by the amount deposited by the user
        assertEq(weth.balanceOf(address(pool)),
            currentBalanceOfWethInPool + minimumDeposit); // ok
    }
```

推奨される軽減策:

- デッドラインに関するロジックを変更するか、削除するか、その使用に適切に機能とパラメータを適応させてください。
- `deadline`ロジックに基づく制限を見直してください。
- デッドライン修飾子を使用して`TSwapPool::deposit`関数を以下の変更を検討できます：

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

[S-高 4] `TSwapPool::getInputAmountBasedOnOutput` の手数料計算が間違っており、0.03% ではなく 10.13% に設定されています。

説明:

戻り値の計算で`10000`が`1000`の代わりに使用されています。

- これは、マジックナンバーを扱う代わりに定数変数名を使用していないことの結果かもしれません。

影響:

- 0.03% ではなく 10.13% を計算します。
- ユーザーは、スワップごとに 0.03% ではなく 10.13% のスワップ額を請求されます。
- スワップを支払う際に過剰に請求されるため、ユーザーは期待された量のトークンを受け取りません。
- ユーザーが支払った余分な金額は、流動性提供者によって引き出されます。

概念実証:

- 以下のテストケースでは、プロトコルのテストスイートTSwapPool.t.solに追加できるものがあります。ここで何が起こっているかです：

1. 流動性提供者は 1:1 の比率でプールに流動性を提供します。
2. 1 トークンを 1Weth にスワップしたいユーザー == スワッパーは、TSwapPool::swapExactOutput関数を呼び出します。
3. スワッパーとプールのスワップ前/後の残高をチェックします。
4. スワッパーは 1weth を得るために 1 ではなく 10.13 トークンを支払いました。
5. 流動性提供者はプールからすべての流動性 + 誤計算からの余分なものを引き出します。

Proof of Code

```
    // test based on getInputAmountBasedOnOutput function and fee
       miscalculation
    function testCalculationFee() public {
        uint256 liquidityFromProvider = 100e18;

        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), liquidityFromProvider);
        poolToken.approve(address(pool), liquidityFromProvider);

        // deposit liquidity into a pool
        //    function deposit(uint256 wethToDeposit, uint256
           minimumLiquidityTokensToMint, uint256
        // maximumPoolTokensToDeposit, uint64 deadline)
        pool.deposit(liquidityFromProvider, 0, liquidityFromProvider,
           uint64(block.timestamp));

        vm.stopPrank(); // end prank liquidity provider

        // liquidity deposited checkers
        assertEq(pool.balanceOf(liquidityProvider),
           liquidityFromProvider);
        console.log("Pool balance of Token A from liquidity provider: "
           , poolToken.balanceOf(address(pool)));
        console.log("Pool balance of Weth from liquidity provider: ",
           weth.balanceOf(address(pool)));

        // start prank user
        vm.startPrank(user); // mint 100e18 in setup == 100 Token A -
           LP / 100 Weth
        uint256 userStartingWethBalance = weth.balanceOf(user);
        uint256 userStartingTokenABalance = poolToken.balanceOf(user);
        console.log("User starting Weth balance: ",
           userStartingWethBalance);
        console.log("User starting Token A balance: ",
           userStartingTokenABalance);
```

```
27
28          // approve token A to be used for swap
29          poolToken.approve(address(pool), type(uint256).max);
30
31          // get 1 Weth in exchange
32          pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.
                timestamp));
33
34          uint256 userEndingWethBalanceAfterSwap = weth.balanceOf(user);
35          uint256 userEndingTokenABalanceAfterSwap = poolToken.balanceOf(
                user);
36
37          vm.stopPrank(); // end prank user
38
39          // Weth balance of Weth should have increase by one
40          assertEq(userEndingWethBalanceAfterSwap,
                userStartingWethBalance + 1e18);
41          console.log("User balance of Weth after swap: ",
                userEndingWethBalanceAfterSwap);
42
43          // As pool initial ratio is 1:1, when buying 1 Weth, 1 Token A
                should be removed from the pool
44          // so user balance of Token A should decrease by 1
45          assertEq(userEndingTokenABalanceAfterSwap,
                userStartingTokenABalance - 1e18);
46          console.log("User balance of Token A after swap: ",
                userEndingTokenABalanceAfterSwap);
47          // for 1 weth
48          // Token A before                100,000000000000000000
49          // Token A Expected after swap    99,000000000000000000
50          // Actual Token A balance after   89,868595686048043120
51          //WHAAAAAT nearly - 10.14 tokens instead of -1 ??
52
53          //so the pool is ...
54          console.log("Pool balance after user swap: ", poolToken.
                balanceOf(address(pool)));
55          // Pool Token A after swap       110,131404313951956880
56          // + 10.13....
57          console.log("Pool Weth after swap ", weth.balanceOf(address(
                pool)));
58          // Pool Weth after swap           99,000000000000000000
59
60          // Give place to the liquidity provider to rug the pool
                including the extra of Token A
61          vm.startPrank(liquidityProvider);
62          pool.withdraw(
63              pool.balanceOf(liquidityProvider),
64              1, // minWethToWithdraw
65              1, // minPoolTokensToWithdraw
66              uint64(block.timestamp)
67          );
```

```
68
69          console.log(
70              "Pool balance of Token A after liquidity provider withdraw
                    all: ", poolToken.balanceOf(address(pool))
71          );
72          console.log("Pool balance of Weth after liquidity provider
                withdraw all: ", weth.balanceOf(address(pool)));
73
74          // new balance of liquidity provider
75          console.log("Liquidity provider balance of Weth after withdraw
                all: ", weth.balanceOf(liquidityProvider));
76          console.log(
77              "Liquidity provider balance of Token A after withdraw all:
                    ", poolToken.balanceOf(liquidityProvider)
78          );
79          // loosing 1 weth but gained 10 token A instead of 1
80
81          assertEq(weth.balanceOf(address(pool)), 0);
82          assertEq(poolToken.balanceOf(address(pool)), 0);
83      }
```

推奨される軽減策:

- 戻り値の計算で`10000`が`1000`に変更してください。

```
1       {
2 -         return ((inputReserves * outputAmount) * 10000) / ((
      outputReserves - outputAmount) * 997);
3 +         return ((inputReserves * outputAmount) * 1_000) / ((
      outputReserves - outputAmount) * 997);
4       }
```

[S-高 5] `TSwapPool::swapExactOutput` にはスリッページ保護がありません。価格の急騰や流動性プール内の資金の大量移動の場合、ユーザーは保護されません。

説明:

価格の急騰や大規模なトランザクションがプールにいつでも発生し得て、`TSwapPool::swapExactInput`には存在するが、`TSwapPool::swapExactOutput`にはスリッページ保護がないため、ユーザーがスリッページに影響を受けます。

瞬間的にユーザーが受け取るべきものの一部分しか支払わなくても良くなる可能性があります。

影響:

- 価格の急騰や大規模なトランザクションがプールにいつでも発生し得て、ユーザーがスリッページに影響を受けます。
- MEV（Miner Extractable Value、マイナー抽出価値）攻撃も発生する可能性があります。

推奨される軽減策:

- 関数宣言に最大出力を確認する追加パラメータと新しいエラーステートメント、そして新しい IF 構文が必要です。

```
+ error TSwapPool__OutputTooBig(uint256 actual, uint256 max);

    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
+       uint256 maxOutputAmount,
        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 (outputAmount > maxOutputAmount) {
+           revert TSwapPool__OutputTooBig(outputAmount,
   maxOutputAmount);
+       }

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

[S-高 6] `TSwapPool::sellPoolTokens` 関数で、ユーザーが間違った量のトークンを受け取ります

説明:

- ビジネスロジックが間違っており、返り値の関数とパラメータが誤っています。
- 返り値で`swapExactOutput`ではなく`swapExactInput`を使用しています。
- 別の報告書の問題で述べられているように、誤った数学とスリッページ保護の欠如により、`Weth` とのトークン交換時にユーザーが大きな損失を被ります。
- 1:1 の比率が守られていません。

影響:

- 上記の誤ったロジックにより、交換時に多額の手数料を請求されるため、ユーザーは期待したトークン量を返されず、莫大な損失を被ります。

概念実証:

- 以下は、`user` が `TSwapPool::sellPoolTokens` 関数を呼び出してプールトークンを Weth に売却するシンプルなテストです。
- 売却するトークンの量は 3 です。
- プロセス中に 3 トークンではなく 32 トークンが消失しました。

```
1     function testSellPoolTokens() public {
2         //result
3         //   amount sold: 3 ether
              3000000000000000000
4         //   User balance of pool token before sell:
              10000000000000000000
5         //   User balance of weth before sell:
              10000000000000000000
6         //   User balance of pool token after sell:
              68979102255219266046  // 32 tokens disappeared instead of 3
7         //   User balance of weth after sell:
              10300000000000000000
8         //   Error: a == b not satisfied [uint]
9         //         Left: 68979102255219266046
10        //        Right: 97000000000000000000
11        uint256 liquidityFromProvider = 100e18;
12        uint256 amountOfTokenToSell = 3 ether;
13
14        vm.startPrank(liquidityProvider);
15        weth.approve(address(pool), liquidityFromProvider);
16        poolToken.approve(address(pool), liquidityFromProvider);
17
```

```
18          pool.deposit(liquidityFromProvider, 0, liquidityFromProvider,
                uint64(block.timestamp));
19
20          vm.stopPrank();
21
22          // Use another user trying to sell pool tokens for weth
23          vm.startPrank(user);
24          uint256 poolTokenBalanceBeforeSell = poolToken.balanceOf(user);
25          uint256 wethBalanceBeforeSell = weth.balanceOf(user);
26          console.log("User balance of pool token before sell: ",
                poolTokenBalanceBeforeSell);
27          console.log("User balance of weth before sell: ",
                wethBalanceBeforeSell);
28
29          poolToken.approve(address(pool), liquidityFromProvider);
30          pool.sellPoolTokens(amountOfTokenToSell);
31
32          uint256 poolTokenBalanceAfterSell = poolToken.balanceOf(user);
33          uint256 wethBalanceAfterSell = weth.balanceOf(user);
34          console.log("User balance of pool token after sell: ",
                poolTokenBalanceAfterSell);
35          console.log("User balance of weth after sell: ",
                wethBalanceAfterSell);
36
37          vm.stopPrank();
38
39          assertLt(poolTokenBalanceAfterSell, poolTokenBalanceBeforeSell)
                ; // ok
40          assertGt(wethBalanceAfterSell, wethBalanceBeforeSell); // ok
41          assertEq(wethBalanceAfterSell, wethBalanceBeforeSell +
                amountOfTokenToSell); // ok
42
43          //The below fail, 32 tokens disappeared... for 1 weth:
44          // ├─ emit log_named_uint(key: "      Left", val:
                6897910225521926604 6 [6.897e19])
45          // ├─ emit log_named_uint(key: "     Right", val:
                9700000000000000000 [9.7e19])
46          assertEq(poolTokenBalanceAfterSell, poolTokenBalanceBeforeSell
                - amountOfTokenToSell); // not ok
47      }
```

推奨される軽減策:

- この報告書の他の関数の問題を確認し、以下の推奨事項に合わせて調整してください：
  - 返り値の文で使用される関数を変更してください：

```
1      function sellPoolTokens(uint256 poolTokenAmount) external  returns
           (uint256 wethAmount) {
2  -       return swapExactOutput(i_poolToken, i_wethToken,
       poolTokenAmount, uint64(block.timestamp));
3  +       return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken
       , poolWethAmount, uint64(block.timestamp));
4      }
```

中

[S-中 1] `ERC721::_mint()` の使用は危険です。`_safeMint()` に置き換え、openzeppelin からの `nonreentrant` ガードを使用してください。

説明:

- `TSwapPool::_addLiquidityMintAndTransfer` で `ERC721::_mint()` を使用すると、ERC721 トークンを ERC721 トークンをサポートしていないアドレスに発行できます。その場合、トークンは永遠に取り出せなくなります。

影響:

- `ERC721::_mint()` を使用すると、ERC721 トークンを ERC721 トークンをサポートしていないアドレスに発行できます（所有していても ERC721 を使用できません）。
- `_safeMint()` は悪意を持って使用され、再入可能性攻撃を許可することもあります。これを防ぐために、openzeppelin からの `nonreentrant` ガードを使用できます。

推奨される軽減策:

- ERC721 トークンの受信者が ERC721 トークンを処理できるコントラクトであり、それらを使用できることを確認するために、`_mint()` の代わりに `_safeMint()` を使用してください。
- 悪意のあるコントラクトが `_safeMint()` を使用してプロトコルを攻撃しようとする場合の再入可能性攻撃を防ぐために、openzeppelin からの修飾子 `nonReentrant` を再入ガードとして使用した方が良いです。

https://docs.openzeppelin.com/contracts/5.x/api/utils#ReentrancyGuard

以下の変更をお勧めします：

TSwapPool::_addLiquidityMintAndTransfer:

```
+ import {ReentrancyGuard} from "@openzeppelin/contracts/utils/
    ReentrancyGuard.sol";

- contract TSwapPool is ERC20
+ contract TSwapPool is ERC20, ReentrancyGuard
.
.
.
    function _addLiquidityMintAndTransfer(
        uint256 wethToDeposit,
        uint256 poolTokensToDeposit,
        uint256 liquidityTokensToMint
    )
        private
+       nonReentrant
    {
-        _mint(msg.sender, liquidityTokensToMint);
+        _safeMint(msg.sender, liquidityTokensToMint);

.
,
,
    }
```

[S-中 2] 転送時の手数料や再配分ロジックを持つ変な ERC20 や ERC777 トークンが、追加トークンの発行や再帰攻撃により不変性を破壊する可能性があります。

説明:

再配分が行われたり、転送時に手数料が発生する変なトークンは、プロトコルの不変性を破壊する可能性があります。再帰問題を抱えるトークンも存在します。

複利で増加するトークンや時間とともに数が増えるトークンは、WETH の数が時間とともに増えないため、プールの比率を破壊します。したがって、1:1 の比率は破られます。

変な ERC20 についての情報：https://github.com/d-xo/weird-erc20

影響:

- プール比率の破壊や、様々なトークンに対する再帰攻撃の可能性。
- コントラクトで使用されている `SafeErc20` ライブラリは、これらのトークンのすべてを保護出来ない可能性があります。

概念実証:

基本的な「転送時手数料」ロジックを使用する変則的なトークンを使用した例：

- 手数料は数値 `1000` に設定されます。したがって、以下の結果の最後の桁も注意して見てください。最初の桁だけを見ないでください。

結果:

```
1 //   Liquidity provider balance of shitcoin:
    10000000000000000000
2 //   User balance of shitcoin:
    999999999999999000
3 //   Liquidity provider balance of shitcoin after transfer to User:
    990000000000000000
4 //   Balance of Weth in the pool:
    100000000000000000
5 //   Balance of WeirdERC20 with fee on transfer:
    999999999999999000
6 //   Balance of Liquidity provider after deposit:
    980000000000000000
7 //   Balance of Liquidity provider after withdraw:
    989999999999998000
8 //   Balance of shitcoin in the pool after withdraw:
                                        0
```

上記の結果のために取られたステップ：

1. トークンを作成し、`liquidity provider`にミントします。-> `10000000000000000000` トークン。
2. `1e18` 相当のトークンを `user` に転送します。-> より簡単に示すために、`user`が期待したトークン量を得ていないことを示します。したがって、`user`が流動性プールである場合も同様です。-> 受け取った `999999999999999000` トークン。
3. 転送後の `liquidity provider` の残高を確認します。-> `990000000000000000` トークン。
4. 1:1 の比率でトークンを預ける流動性プールを作成します。
5. 預金後のプールの残高を確認 -> `999999999999999000` トークン。
6. 預金後の `liquidity provider` の残高 -> `980000000000000000` トークン。
7. プールからすべての流動性を引き出します。

8. 引き出し後の `liquidity provider` の残高を確認 -> `98999999999999998000` トークン。

手数料のため、転送が発生するプロセスでは、トークン量が `1000` ずつ減少しますが、`WETH` はデフォルトで減少しませんので、比率が破綻します。

- 上記のテストに必要な要件：`./test/mocks/` に以下のコードを含む `ERC20MockFeeOnTransfer.sol` という追加のファイルを配置する必要があります

ERC20MockFeeOnTransfer code

```
1
2  // Copyright (C) 2017, 2018, 2019, 2020 dbrock, rain, mrchico, d-xo
3  // SPDX-License-Identifier: AGPL-3.0-only
4
5  // adapted from https://github.com/d-xo/weird-erc20/blob/main/src/
       TransferFee.sol
6
7  pragma solidity >=0.6.12;
8
9  contract Math {
10 // --- Math ---
11 function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
12 require((z = x + y) >= x);
13 }
14
15     function sub(uint256 x, uint256 y) internal pure returns (uint256 z
           ) {
16         require((z = x - y) <= x);
17     }
18
19 }
20
21 contract WeirdERC20 is Math {
22 // --- ERC20 Data ---
23 string public name;
24 string public symbol;
25 uint8 public decimals;
26 uint256 public totalSupply;
27 bool internal allowMint = true;
28
29     mapping(address => uint256) public balanceOf;
30     mapping(address => mapping(address => uint256)) public allowance;
31
32     event Approval(address indexed src, address indexed guy, uint256
           wad);
33     event Transfer(address indexed src, address indexed dst, uint256
           wad);
34
35     // --- Init ---
```

```
36      constructor(string memory _name, string memory _symbol, uint8
            _decimalPlaces) public {
37          name = _name;
38          symbol = _symbol;
39          decimals = _decimalPlaces;
40      }
41
42      // --- Token ---
43      function transfer(address dst, uint256 wad) public virtual returns
            (bool) {
44          return transferFrom(msg.sender, dst, wad);
45      }
46
47      function transferFrom(address src, address dst, uint256 wad) public
             virtual returns (bool) {
48          require(balanceOf[src] >= wad, "WeirdERC20: insufficient-
                balance");
49          if (src != msg.sender && allowance[src][msg.sender] != type(
                uint256).max) {
50              require(allowance[src][msg.sender] >= wad, "WeirdERC20:
                    insufficient-allowance");
51              allowance[src][msg.sender] = sub(allowance[src][msg.sender
                    ], wad);
52          }
53          balanceOf[src] = sub(balanceOf[src], wad);
54          balanceOf[dst] = add(balanceOf[dst], wad);
55          emit Transfer(src, dst, wad);
56          return true;
57      }
58
59      function approve(address usr, uint256 wad) public virtual returns (
            bool) {
60          allowance[msg.sender][usr] = wad;
61          emit Approval(msg.sender, usr, wad);
62          return true;
63      }
64
65      function mint(address to, uint256 _amount) public {
66          require(allowMint, "WeirdERC20: minting is off");
67
68          _mint(to, _amount);
69      }
70
71      function _mint(address account, uint256 amount) internal virtual {
72          require(account != address(0), "WeirdERC20: mint to the zero
                address");
73
74          totalSupply += amount;
75          unchecked {
76              // Overflow not possible: balance + amount is at most
                    totalSupply + amount, which is checked above.
```

```
77              balanceOf[account] += amount;
78          }
79          emit Transfer(address(0), account, amount);
80      }
81
82      function burn(address from, uint256 _amount) public {
83          _burn(from, _amount);
84      }
85
86      function _burn(address account, uint256 amount) internal virtual {
87          require(account != address(0), "WeirdERC20: burn from the zero
                address");
88
89          uint256 accountBalance = balanceOf[account];
90          require(accountBalance >= amount, "WeirdERC20: burn amount
                exceeds balance");
91          unchecked {
92              balanceOf[account] = accountBalance - amount;
93              // Overflow not possible: amount <= accountBalance <=
                    totalSupply.
94              totalSupply -= amount;
95          }
96
97          emit Transfer(account, address(0), amount);
98      }
99
100     function toggleMint() public {
101         allowMint = !allowMint;
102     }
103
104 }
105
106 contract ERC20MockFeeOnTransfer is WeirdERC20 {
107 uint256 private fee;
108
109     // --- Init ---
110     constructor(
111         string memory _name,
112         string memory _symbol,
113         uint8 _decimalPlaces,
114         uint256 _fee
115     )
116         WeirdERC20(_name, _symbol, _decimalPlaces)
117     {
118         fee = _fee;
119     }
120
121     // --- Token ---
122     function transferFrom(address src, address dst, uint256 wad) public
            override returns (bool) {
123         require(balanceOf[src] >= wad, "ERC20MockFeeOnTransfer:
```

```
            insufficient-balance");
124         // don't worry about allowances for this mock
125         //if (src != msg.sender && allowance[src][msg.sender] != type(
            uint).max) {
126         //    require(allowance[src][msg.sender] >= wad, "
            ERC20MockFeeOnTransfer insufficient-allowance");
127         //    allowance[src][msg.sender] = sub(allowance[src][msg.
            sender], wad);
128         //}
129
130         balanceOf[src] = sub(balanceOf[src], wad);
131         balanceOf[dst] = add(balanceOf[dst], sub(wad, fee));
132         balanceOf[address(0)] = add(balanceOf[address(0)], fee);
133
134         emit Transfer(src, dst, sub(wad, fee));
135         emit Transfer(src, address(0), fee);
136
137         return true;
138     }
139
140 }
```

- TSwapPool.t.sol にインポートとコントラクト宣言/作成を追加します、以下を参照：

```
1  import { Test, console } from "forge-std/Test.sol";
2  import { TSwapPool } from "../../src/PoolFactory.sol";
3  import { ERC20Mock } from "@openzeppelin/contracts/mocks/token/
       ERC20Mock.sol";
4  import { IERC20 } from "@openzeppelin/contracts/interfaces/IERC20.sol";
5  + import { ERC20MockFeeOnTransfer } from "../mocks/
       ERC20MockFeeOnTransfer.sol"; // ERC20MockFeeOnTransfer.sol for weird
6      // token test
7
8  contract TSwapPoolTest is Test {
9      TSwapPool pool;
10     ERC20Mock poolToken;
11     ERC20Mock weth;
12 +     ERC20MockFeeOnTransfer shitcoin; // ERC20MockFeeOnTransfer.sol for
       weird token test
13
14     address liquidityProvider = makeAddr("liquidityProvider");
15     address user = makeAddr("user");
16
17     function setUp() public {
18         poolToken = new ERC20Mock();
19         weth = new ERC20Mock();
20
21 +         shitcoin = new ERC20MockFeeOnTransfer("FeeOnTransferCoin", "
       SHIT", 18, 1000); // ERC20MockFeeOnTransfer.sol
22             // for
23             // weird token test || parameters : name, symbol, decimals
```

```
                    number, fee when transfer occur
24
25          pool = new TSwapPool(address(poolToken), address(weth), "
                LTokenA", "LA");
26
27          weth.mint(liquidityProvider, 200e18);
28          poolToken.mint(liquidityProvider, 200e18);
29
30          weth.mint(user, 100e18);
31          poolToken.mint(user, 100e18);
32      }
33  .
34  .
35  .
36  }
```

- 次に、`TSwapPool.t.sol` に以下のテストケースを追加します：

Test code

```
1       function testWeirdErc20WithFee() public {
2           // fee set to 1000 at contract creation in setup
3
4           //   Liquidity provider balance of shitcoin:
                                    100000000000000000000
5           //   User balance of shitcoin:
                                                    999999999999999000
6           //   Liquidity provider balance of shitcoin after transfer to
                User:  990000000000000000
7           //   Balance of Weth in the pool:
                                                  100000000000000000
8           //   Balance of WeirdERC20 with fee on transfer:
                                     999999999999999000
9           //   Balance of Liquidity provider after deposit:
                                  980000000000000000
10          //   Balance of Liquidity provider after withdraw:
                                 989999999999998000
11          //   Balance of shitcoin in the pool after withdraw:
                                                  0
12          uint256 amountUsedForInteraction = 1 ether;
13
14          vm.startPrank(liquidityProvider);
15          shitcoin.mint(address(liquidityProvider), 100e18);
16          // balance of shitcoin in liquidity provider
17          uint256 startingShitcoinBalance = shitcoin.balanceOf(
                liquidityProvider);
18          console.log("Liquidity provider balance of shitcoin: ",
                startingShitcoinBalance);
19
20          // transfer shitcoin token to user and check user balance,
                transfer 2e18 (2 ether value)
```

```
21          shitcoin.transferFrom(address(liquidityProvider), address(user)
                , amountUsedForInteraction);
22          console.log("User balance of shitcoin: ", shitcoin.balanceOf(
                user));
23          // 1 ether instead of 2 has been transferred, due to 1 ether
                fee.
24
25          // check liquidity provider after transfer
26          console.log(
27              "Liquidity provider balance of shitcoin after transfer to
                    User: ", shitcoin.balanceOf(liquidityProvider)
28          );
29
30          // now let deposit into a pool and withdraw from the pool
31          // create the pool
32          pool = new TSwapPool(address(shitcoin), address(weth), "SHIT/
                WETH", "SW");
33          // approve token
34          weth.approve(address(pool), amountUsedForInteraction);
35          shitcoin.approve(address(pool), amountUsedForInteraction);
36
37          //capture balance before deposit, as in this test a transfer to
                 external user is also made, for easy
38          // understanding.
39          uint256 balanceBeforeDeposit = shitcoin.balanceOf(address(
                liquidityProvider));
40
41          //deposit to the pool
42          pool.deposit(amountUsedForInteraction, 0,
                amountUsedForInteraction, uint64(block.timestamp));
43
44          // check deposited tokens
45          console.log("Balance of Weth in the pool: ", weth.balanceOf(
                address(pool)));
46          console.log("Balance of WeirdERC20 with fee on transfer: ",
                shitcoin.balanceOf(address(pool)));
47
48          // capture the actual balance of shitcoin after deposit
49          uint256 shitcoinBalanceAfterDeposit = shitcoin.balanceOf(
                address(liquidityProvider));
50          console.log("Balance of Liquidity provider after deposit: ",
                shitcoinBalanceAfterDeposit);
51
52          // withdraw tokens from the pool
53          pool.withdraw(
54              pool.balanceOf(liquidityProvider),
55              1, // minWethToWithdraw
56              1, // minPoolTokensToWithdraw
57              uint64(block.timestamp)
58          );
59
```

```
60          // check if initial shitcoin balance and ending balance is
                equal
61          uint256 shitcoinBalanceAfterWithdraw = shitcoin.balanceOf(
                liquidityProvider);
62          console.log("Balance of Liquidity provider after withdraw: ",
                shitcoinBalanceAfterWithdraw);
63
64          // pool balance
65          console.log("Balance of shitcoin in the pool after withdraw: ",
                 shitcoin.balanceOf(address(pool)));
66          vm.stopPrank();
67
68          assertLt(shitcoinBalanceAfterWithdraw, balanceBeforeDeposit);
69      }
```

推奨される軽減策:

- プロトコルで使用できるかどうかのトークンのホワイトリスト/ブラックリスト配列を作成する。
- よく知られていないトークンの使用を制限する。
- そのようなトークンは手動でレビューする必要があります。
- トークンがレビューされた場合、他のプール作成のためにホワイトリストに登録する。
- または AAVE プロトコルのように、よく知られているトークンのみを使用する。

低

[S-低 1] `PoolCreated` イベントは、検索性とフィルタリングのために `indexed` にするべきです。

説明:

- AMM であるため、サードパーティアプリがプールの状態を追跡できるように、イベントをインデックス付けすることが望ましいです。
- `PoolFactory` コントラクトはインデックス付けされておらず、特定のトランザクションや状態変化を検索およびフィルタリングする際に困難を生じます。
- `TSwapPool` の主要イベントは、3 つのパラメータを使用して `indexed` されています。
- 注: インデックス付けされたイベントは、より効率的に保存されます。

影響:

- イベントの取得およびフィルタリングが困難です。これは低重大度の問題ですが、検索性とフィルタリングを向上させるためにイベントをインデックス付けすることは良い慣行です。

推奨される軽減策:

`indexed` キーワードを追加します：

```
1 -    event PoolCreated(address tokenAddress, address poolAddress);
2
3 +    event PoolCreated(address tokenAddress, address poolAddress)
     indexed;
```

[S-低 2] 関数シグネチャで宣言されているが `TSwapPool::swapExactInput` 関数内に存在しない戻り値 `uint256 output`

説明:

- 戻り値 `uint256 output` は `TSwapPool::swapExactInput` 関数内に存在しませんが、関数シグネチャで宣言されています。

影響:

- 戻り値は常に 0 になります。
- 影響は低いですが、値を出力するロジックがある場合、関数内で宣言されるべきです。

推奨される軽減策:

- 戻り値として `outputAmount` を使用するべきだと思いますが、その場合、関数自体内で設定されて戻り値として使用される必要があります。

[S-低 3] `TSwapPool::LiquidityAdded` イベントのパラメータが誤った順序で、返される情報が誤っています。

説明:

`uint256 poolTokensDeposited` と `uint256 wethDeposited` が逆になっています。

- 返される情報が誤っています。

発行されるイベント：

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

宣言されたイベント：

```
1  -    event LiquidityAdded(address indexed liquidityProvider, uint256
   wethDeposited, uint256 poolTokensDeposited);
2  +    event LiquidityAdded(address indexed liquidityProvider, uint256
   poolTokensDeposited, uint256 wethDeposited);
```

情報系

[S-情報系 1] マジックナンバー

説明:

すべての数値リテラルは定数に置き換えるべきです。これにより、コードの可読性が向上し、保守が容易になります。文脈なしの数値は「マジックナンバー」と呼ばれます。

推奨される軽減策:

- すべてのマジックナンバーを定数に置き換えてください。

TSwapPool.sol コントラクト:

```
+       uint256 public constant WHATEVER_IS_997 = 997;
+       uint256 public constant WHATEVER_IS_1000 = 1000;
+       uint256 public constant WHATEVER_IS_10000 = 10000;
+       uint256 public constant WHATEVER_IS_1_000_000_000_000_000_000 =
 1_000_000_000_000_000_000;
+       uint256 public constant WHATEVER_IS_1e18 = 1e18;
.
.
.
-       uint256 inputAmountMinusFee = inputAmount * 997;
-       uint256 denominator = (inputReserves * 1000) +
inputAmountMinusFee;
+       uint256 inputAmountMinusFee = inputAmount * WHATEVER_IS_997;
+       uint256 denominator = (inputReserves * WHATEVER_IS_1000) +
inputAmountMinusFee;
.
.
.
-        return ((inputReserves * outputAmount) * 10000) / ((
outputReserves - outputAmount) * 997);
+        return ((inputReserves * outputAmount) * WHATEVER_IS_10000) /
((outputReserves - outputAmount) * WHATEVER_IS_997);
.
.
.
-       outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000
);
+       outputToken.safeTransfer(msg.sender,
WHATEVER_IS_1_000_000_000_000_000_000);
.
.
.
-       getOutputAmountBasedOnInput(1e18, i_wethToken.balanceOf(
address(this)), i_poolToken.balanceOf(address(this)));
```

```
27 +         getOutputAmountBasedOnInput(WHATEVER_IS_1e18, i_wethToken.
       balanceOf(address(this)), i_poolToken.balanceOf(address(this)));
28 .
29 .
30 .
31 -         getOutputAmountBasedOnInput(1e18, i_poolToken.balanceOf(
       address(this)), i_wethToken.balanceOf(address(this)));
32 +         getOutputAmountBasedOnInput(WHATEVER_IS_1e18, i_poolToken.
       balanceOf(address(this)), i_wethToken.balanceOf(address(this)));
```

[S-情報系 2] TSwapPool コントラクトの swapExactInput() 関数は使用されていません。削除するか、その使用法を変更してください。

説明:

swapExactInput() 関数は使用されておらず、削除されるか、external として使用されるように変更するべきです。または、その使用法のロジックを、意図された用途に応じて変更してください。

```
1     function swapExactInput(
2         IERC20 inputToken,
3         uint256 inputAmount,
4         IERC20 outputToken,
5         uint256 minOutputAmount,
6         uint64 deadline
7     )
8 -       public
9 +       external
10        revertIfZero(inputAmount)
11        revertIfDeadlinePassed(deadline)
12        returns (uint256 output)
13    {
14        uint256 inputReserves = inputToken.balanceOf(address(this));
15        uint256 outputReserves = outputToken.balanceOf(address(this));
16
17        uint256 outputAmount = getOutputAmountBasedOnInput(
18            inputAmount,
19            inputReserves,
20            outputReserves
21        );
22
23        if (outputAmount < minOutputAmount) {
24            revert TSwapPool__OutputTooLow(outputAmount,
                  minOutputAmount);
25        }
```

```
26
27          _swap(inputToken, inputAmount, outputToken, outputAmount);
28      }
```

[S-情報系 3] Solidity 0.8.20 はデフォルトで EVM バージョンを Shanghai に対象としており、PUSH0 はすべてのチェーンでサポートされていません。

説明:

Solc コンパイラバージョン 0.8.20 は、デフォルトのターゲット EVM バージョンを Shanghai に切り替えるため、生成されたバイトコードには PUSH0 オペコードが含まれます。メインネット以外のチェーン、特に PUSH0 をサポートしていない L2 チェーンにデプロイする場合は、適切な EVM バージョンを選択してください。そうしないと、コントラクトのデプロイメントが失敗します。

- このプロトコルは Ethereum にデプロイされることを意図しているため、問題ではありませんが、念のための情報です。

[S-情報系 4] `TSwapPool::deposit` 関数で変数 `poolTokenReserves` は使用されていません。削除するか、その使用法を変更してください。

説明:

`TSwapPool::deposit` 関数で変数 `poolTokenReserves` は使用されておらず、削除されるか、下記のコメントで説明されているロジックに従って統合されるか、その使用法のロジックを、意図された用途に応じて変更してください。

```
1           if (totalLiquidityTokenSupply() > 0) {
2               uint256 wethReserves = i_wethToken.balanceOf(address(this))
                    ;
3   -           uint256 poolTokenReserves = i_poolToken.balanceOf(address(
        this)); // @audit unused local variable
4               uint256 poolTokensToDeposit =
                    getPoolTokensToDepositBasedOnWeth(
5                   wethToDeposit
6               );
7               if (maximumPoolTokensToDeposit < poolTokensToDeposit) {
8                   revert TSwapPool__MaxPoolTokenDepositTooHigh(
```

```
                    maximumPoolTokensToDeposit,
                    poolTokensToDeposit
                );
            }
            liquidityTokensToMint =
                (wethToDeposit * totalLiquidityTokenSupply()) /
                wethReserves;
            if (liquidityTokensToMint < minimumLiquidityTokensToMint) {
                revert TSwapPool__MinLiquidityTokensToMintTooLow(
                    minimumLiquidityTokensToMint,
                    liquidityTokensToMint
                );
            }
            _addLiquidityMintAndTransfer(
                wethToDeposit,
                poolTokensToDeposit,
                liquidityTokensToMint
            );
        }
```

[S-情報系 5] コンストラクターでのゼロチェックが不足しています。

説明:

`PoolFactory::constructor` と `TSwapPool::constructor` の `wethToken` パラメーターでゼロチェックが不足しています。

`TSwapPool::constructor`:

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+       if (poolToken == address(0) || wethToken == address(0)) {
+           revert TSwapPool__InvalidToken();
*       }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

PoolFactory::constructor:

```
+   error PoolFactory__ZeroAddress();

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

[S-情報系 6] IERC20 インターフェースが重複しており、同じ機能が最新バージョンの solidity を使用する OpenZeppelin の `ERC20.sol` で利用可能です。

説明:

TSwapPool コントラクトで OpenZeppelin から、PoolFactory コントラクトで Forge-Std ライブラリから 2 つの IERC20 インターフェースをインポートしています。OpenZeppelin からのものが十分であり、使用されるべきです。

PoolFactory::IERC20、以下の関数は OpenZeppelin の ERC20.sol ライブラリで利用可能です。

```
    /// @notice Returns the name of the token.
    function name() external view returns (string memory);

    /// @notice Returns the symbol of the token.
    function symbol() external view returns (string memory);

    /// @notice Returns the decimals places of the token.
    function decimals() external view returns (uint8);
```

PoolFactory コントラクトで name()、symbol()、decimals() を使用される場所で変更可能です。

- また、Forge-Std からのものは、プロジェクトで使用されている他のライブラリより異なる solidity バージョンを使用しています。

```
    - [0.8.20](src/PoolFactory.sol#L15)
    - [0.8.20](src/TSwapPool.sol#L15)
-   - [>=0.6.2](lib/forge-std/src/interfaces/IERC20.sol#L2)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/interfaces/draft-
      IERC6093.sol#L3)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.
      sol#L4)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/token/ERC20/IERC20
      .sol#L4)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/token/ERC20/
      extensions/IERC20Metadata.sol#L4)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/token/ERC20/
      extensions/IERC20Permit.sol#L4)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/token/ERC20/utils/
      SafeERC20.sol#L4)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/utils/Address.sol#
      L4)
    - [^0.8.20](lib/openzeppelin-contracts/contracts/utils/Context.sol#
      L4)
```

[S-情報系 7] トークンシンボルを取得するために間違った関数が使用されています。

説明:

PoolFactory::createPool 関数で、トークンシンボルの文字列を結合する際に .name() ではなく .symbol() を使用するべきです。

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

PoolFactory コントラクトで name()、symbol()、decimals() を使用される場所で変更可能です。

[S-情報系 8] `PoolFactory::PoolFactory__PoolDoesNotExist` 未使用のエラー。

説明:

`PoolFactory` コントラクトで宣言されている未使用のエラー。`PoolFactory::PoolFactory__PoolDoesNotExist`

```
1     error PoolFactory__PoolDoesNotExist();
```

[S-情報系 9] 外部コールの前に“状態変更” が行うべき、状態変数でなくても CEI に従って設定すべきです。

説明:

`TSwapPool::deposit` 関数の else 文で、`_addLiquidityMintAndTransfer` 関数コールの前に `liquidityTokensToMint` 変数を設定するべきです。これは、Checks-Effects-Interactions パターンに従うためです。このケースでは再入可能性の危険はありませんが、良い習慣です。

```
1  else {
2              // This will be the "initial" funding of the protocol. We
                  are starting from blank here!
3              // We just have them send the tokens in, and we mint
                  liquidity tokens based on the weth
4  +           liquidityTokensToMint = wethToDeposit;
5              _addLiquidityMintAndTransfer(wethToDeposit,
                  maximumPoolTokensToDeposit, wethToDeposit);
6  -           liquidityTokensToMint = wethToDeposit;
7          }
```

[S-情報系 10] `TSwapPool::totalLiquidityTokenSupply` 関数は `public` ではなく `external` であるべきです。

説明:

`TSwapPool::totalLiquidityTokenSupply` 関数は **`public`** ではなく `external` であるべきです。

```
-   function totalLiquidityTokenSupply() public view returns (uint256) {
+   function totalLiquidityTokenSupply() external view returns (uint256) {
        return totalSupply();
    }
```

ガス

- ガスについては、上記の情報の中で含まれている。