Competitive Security Assessment

EchoDEX

Jul 13th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	EchoDEX
Platform & Language	Solidity
Codebase	• https://github.com/echodex-io/echodex-contracts/tree/main/contracts • audit commit - 0c6fe46ce91528467d60057f4f32c65a99054ace • final commit - 8f0c5913cd0fc1d0eb34e9b41cce50de63ec3a3f
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

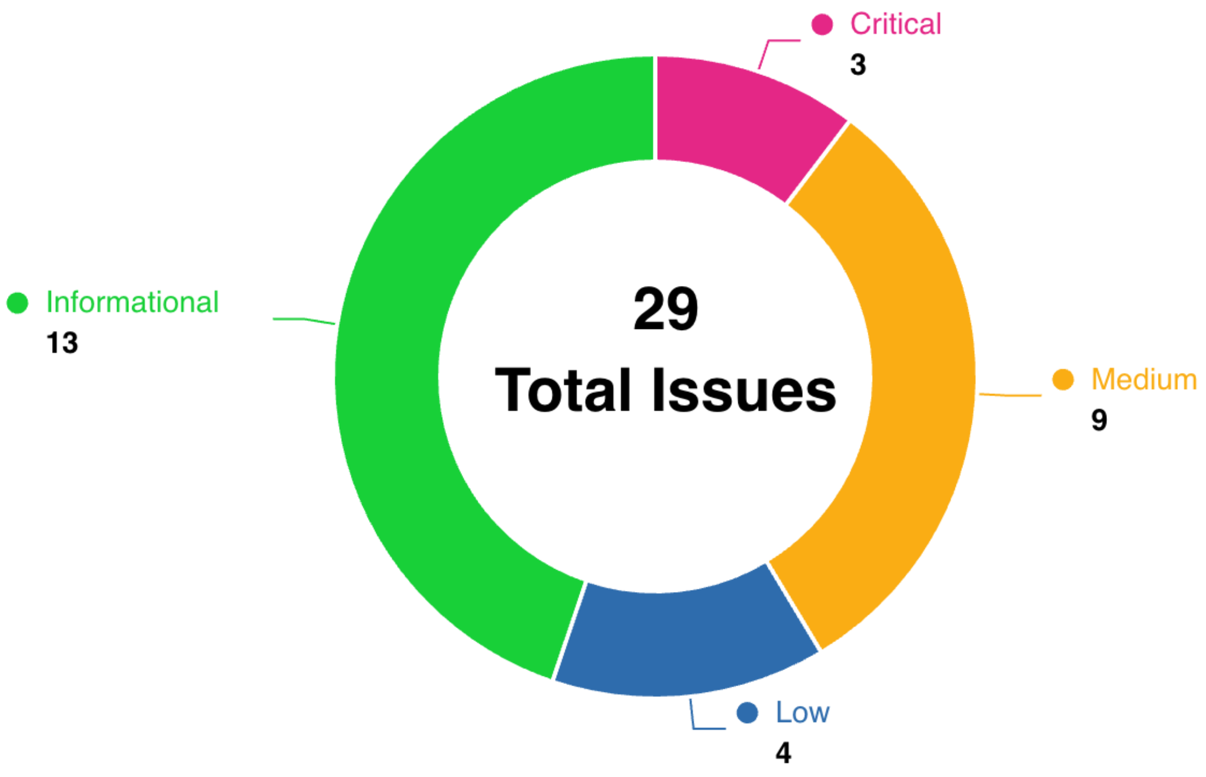
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**3**	**0**	**1**	**1**	**1**	**0**
Medium	**9**	**0**	**1**	**7**	**1**	**0**
Low	**4**	**0**	**2**	**1**	**1**	**0**
Informational	**13**	**0**	**2**	**10**	**0**	**1**

Audit Scope

File	Commit Hash
contracts/EchodexRouterFee.sol	0c6fe46ce91528467d60057f4f32c65a99054ace
contracts/EchodexPair.sol	0c6fe46ce91528467d60057f4f32c65a99054ace
contracts/EchodexFarm.sol	0c6fe46ce91528467d60057f4f32c65a99054ace
contracts/libraries/EchodexLibrary.sol	0c6fe46ce91528467d60057f4f32c65a99054ace
contracts/EchodexFactory.sol	0c6fe46ce91528467d60057f4f32c65a99054ace

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
ECD-1	Need to check whether `tokenFee` is different from `token0` and `token1` in `EchodexPair`	Logical	Critical	Acknowledged	biakia
ECD-2	Potential reward over-distribution issue	Logical	Critical	Fixed	biakia
ECD-3	`amountsFeeAddMore` processing logic errors, which may cause asset loss or swap failure	Logical	Critical	Mitigated	Xi_Zi
ECD-4	old `tokenFee` will be locked in the pair after `tokenFee` is updated	Logical	Medium	Fixed	biakia

ECD-5	Possible failure of the call `withdrawExcessReward` function	Logical	Medium	Fixed	biakia
ECD-6	Incorrect input parameters for flash swap	Logical	Medium	Fixed	biakia
ECD-7	Use `safeTransfer` instead of `transfer`	Logical	Medium	Fixed	ginlee, ladboy233
ECD-8	Potential reentrancy risk in `EchodexFarm::harvest()`	Reentrancy	Medium	Fixed	biakia, ginlee, ladboy233
ECD-9	Missing `nonce` and `chain_id` in `EchodexRouterFee` contract, `removeLiquidityWithPermit` function may lead to signature replay attacks	Signature Forgery or Replay	Medium	Acknowledged	ginlee
ECD-10	Need input validation for `stake` function in `EchodexFarm` contract	Logical	Medium	Fixed	ginlee
ECD-11	Use spot price as fee amount oracle is vulnerable for manipulation	Oracle Manipulation	Medium	Mitigated	ladboy233
ECD-12	The `amountFeeRefundTokenOut` calculation precision error in the `calcFee` function	Logical	Medium	Fixed	Xi_Zi
ECD-13	Input validation in `EchodexLibrary` contract `pairFor` function	Logical	Low	Acknowledged	ginlee
ECD-14	Incompatibility With Deflationary Tokens	Logical	Low	Mitigated	biakia, ladboy233
ECD-15	Unchecked ERC-20 `transferFrom()` Call	Logical	Low	Acknowledged	biakia
ECD-16	Swap x * y = k invariant does not hold	Logical	Low	Fixed	ladboy233
ECD-17	Unlocked Pragma Version	Language Specific	Informational	Fixed	biakia, ginlee
ECD-18	Remove SafeMath library if use solidity version no less than 0.8	Language Specific	Informational	Acknowledged	ginlee
ECD-19	Consider using custom errors instead of string error message	Language Specific	Informational	Acknowledged	ginlee

ECD-20	Gas Optimization: Unnecessary value set to 0 in `EchodexPair.sol`	Gas Optimization	Informational	Fixed	biakia
ECD-21	Gas Optimization :Unused variable in `EchodexFactory.sol` and `EchodexPair.sol`	Gas Optimization	Informational	Fixed	Xi_Zi, biakia
ECD-22	Lack of reasonable upper boundary in `setRefundPercentPair()`	Logical	Informational	Fixed	biakia
ECD-23	`Initialize` function may not suitable in `EchodexPair` contract	Logical	Informational	Declined	ginlee
ECD-24	Gas Optimization: No need to call `abi.encodePacked` when there's only a single bytes argument	Gas Optimization	Informational	Fixed	Xi_Zi
ECD-25	Variables that could be declared as immutable	Language Specific	Informational	Fixed	biakia
ECD-26	Events is missing indexed fields	Language Specific	Informational	Fixed	ginlee
ECD-27	Gas Optimization: Remove unused imports in `EchodexLibrary`	Gas Optimization	Informational	Fixed	biakia
ECD-28	Failure to perform zero-address checks may result in the failure of function execution.	Logical	Informational	Fixed	Xi_Zi
ECD-29	Gas Optimization: Redundant code in `EchodexLibrary.sol`	Code Style	Informational	Fixed	Xi_Zi

ECD-1:Need to check whether `tokenFee` is different from `token0` and `token1` in `EchodexPair`

Category	Severity	Status	Contributor
Logical	Critical	Acknowledged	biakia

Code Reference

- code/contracts/EchodexPair.sol#L205-L228
- code/contracts/EchodexPair.sol#L230-L237

```solidity
    function swapPayWithTokenFee(uint amount0Out, uint amount1Out, address to, address refundFeeAddress, bytes calldata data) external lock { // payWithTokenFee = true
        SwapState memory state = _preSwap(amount0Out, amount1Out, to);

        uint amountOut = amount0Out > 0 ? amount0Out : amount1Out;
        address tokenOut = amount0Out > 0 ? token0 : token1;

        //fee
        (uint fee, uint feeRefund) = IEchodexFactory(factory).calcFee(amountOut, tokenOut, address(this), factory);
        _payFee(fee, feeRefund, refundFeeAddress);
        _safeTransfer(tokenOut, to, amountOut);

        if (data.length > 0) IEchodexCallee(to).echodexCall(msg.sender, amount0Out, amount1Out, data);
        state.balance0 = IERC20(token0).balanceOf(address(this));
        state.balance1 = IERC20(token1).balanceOf(address(this));

        state.amount0In = state.balance0 > state._reserve0 - amount0Out ? state.balance0 - (state._reserve0 - amount0Out) : 0;
        state.amount1In = state.balance1 > state._reserve1 - amount1Out ? state.balance1 - (state._reserve1 - amount1Out) : 0;
        require(state.amount0In > 0 || state.amount1In > 0, 'Echodex: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reserve1), 'Echodex: K');
        }
        _update(state.balance0, state.balance1, state._reserve0, state._reserve1);
        emit Swap(msg.sender, state.amount0In, state.amount1In, amount0Out, amount1Out, to, fee, feeRefund);
    }

    function addFee(uint amount) external lock {
        address tokenFee = IEchodexFactory(factory).tokenFee();
        IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);
        totalFee = totalFee + amount;
        currentFee = currentFee + amount;

        emit AddFee(amount);
    }
```

Description

biakia : The `swapPayWithTokenFee` function is a new way to swap tokens. It will use the `tokenFee` instead of `token0` or `token1` to pay the fee:

```
        //fee
        (uint fee, uint feeRefund) = IEchodexFactory(factory).calcFee(amountOut, tokenOut, address(this), factory);
        _payFee(fee, feeRefund, refundFeeAddress);
        _safeTransfer(tokenOut, to, amountOut);
```

However, there is no check on whether `tokenFee` is different from `token0` and `token1`. If the `tokenFee` is the same with `token0` or `token1`, it is possible that the `swapPayWithTokenFee` function will fail to be called. Let's say the balance of `token0` is 10000 and the balance of `token1` is 10000. So the `reserve0` is 10000 and `reserve1` is 10000. The `tokenFee` is the same with `token0`. The `currentFee` is 100. Now, Alice calls `swapPayWithTokenFee` function to pay 1000 `token1` to get `token0`. The `amountOut` will be 909 and Let's say we have 1% fee for each swap and no refund fee. So the `fee` is calculated as 9.09 and `feeRefund` is calculated as 0. In function `_payFee`, some of `tokenFee` will be sent to `receiveFeeAddress` and `refundFeeAddress`:

```
function _payFee(uint fee, uint feeRefund, address refundFeeAddress) private { //payWithTokenFee = true
        address tokenFee = IEchodexFactory(factory).tokenFee();
        address receiveFeeAddress = IEchodexFactory(factory).receiveFeeAddress();
        require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');

        currentFee = currentFee - fee;
        _safeTransfer(tokenFee, receiveFeeAddress, fee - feeRefund);
        if (feeRefund > 0) {
            _safeTransfer(tokenFee, refundFeeAddress, feeRefund);
        }
    }
```

Since the `tokenFee` is the `token0`, so 9.09 `token0` will be sent to `receiveFeeAddress`. After it, 909 `token0` will be sent to the user:

```
_safeTransfer(tokenOut, to, amountOut);
```

Finally, 918.09(909+9.09) `token0` has been transferred out. Now, the balance of the `token0` is 9181.91(10000+100-918.09) and the balance of the `token1` is 11000. In `swapPayWithTokenFee` function, there is a check on the `K` value:

```
 { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reserve1), 'Echodex: K');
 }
```

Here the `state.balance0` is 9181.91, the `state.balance1` is 11000 and the `state._reserve0` and `state._reserve1` are 10000. The require statement will be `require(9181.91 * 11000>10000 * 10000)` which will be passed. At last, the `_update` function will be called:

```
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
         require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reserve1), 'Echodex: K');
         }
         _update(state.balance0, state.balance1, state._reserve0, state._reserve1);
```

After `_update` is called, the `reserve0` will be 9181.91, the `reserve1` will be 11000 and the `currentFee` will be 100-9.09 = 90.91.

Here we can see that the value of `currentFee` has been added into the `reserve0`.

Now Bob calls `swapPayWithTokenFee` function to pay 1000 `token1` to get `token0`. The `amountOut` will be 765 and then the `fee` is 7.65 and `feeRefund` is still 0. Since the `tokenFee` is the `token0`, so 7.65 `token0` will be sent to `receiveFeeAddress`. After it, 765 `token0` will be sent to the user.

Finally, 772.65(765+7.65) `token0` has been transferred out. Now, the balance of the `token0` is 8409.26(9181.91-772.65) and the balance of the `token1` is 12000.

Now Let's move to the `K` value check, The require statement will be `require(8409.26 * 12000>9181.91 * 11000)`. Since 8409.26 * 12000 is less than 9181.91 * 11000, the require statement will revert.

At last, Bob will fail to call `swapPayWithTokenFee` even though there is enough fee(the `currentFee` is 90.91 now) in the pair.

biakia : In contract `EchodexPair`, the function `addFee` is used to add fee to the pair:

```
   function addFee(uint amount) external lock {
         address tokenFee = IEchodexFactory(factory).tokenFee();
         IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);
         totalFee = totalFee + amount;
         currentFee = currentFee + amount;

         emit AddFee(amount);
     }
```

It will directly transfer `tokenFee` to the pair. The issue here is that it doesn't check if `tokenFee` is different from `token0` and `token1`. If the `tokenFee` is `token0`, after calling `addFee` function, the balance of `token0` will be greater than `reserve0`. Anyone can call `skim` function to steal these added `token0` from the pair:

```
  function skim(address to) external lock {
         address _token0 = token0; // gas savings
         address _token1 = token1; // gas savings
         _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
         _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
     }
```

Recommendation

biakia : Consider making sure the `tokenFee` is different from `token0` and `token1`.

biakia : Consider adding a check in function `addFee`:

```
    function addFee(uint amount) external lock {
            address tokenFee = IEchodexFactory(factory).tokenFee();
            require(tokenFee!=token0 && tokenFee!=token1,"invalid tokenFee");
            IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);
            totalFee = totalFee + amount;
            currentFee = currentFee + amount;

            emit AddFee(amount);
       }
```

Client Response

Acknowledged, tokenFee can be used both for trading and for paying fees.

ECD-2:Potential reward over-distribution issue

Category	Severity	Status	Contributor
Logical	Critical	Fixed	biakia

Code Reference

- code/contracts/EchodexFarm.sol#L227-L239

```
227:    function _update(Pool storage pool) private {
228:        if (pool.totalLP > 0 && pool.lastRewardTimestamp <= pool.endDate) {
229:            uint256 currentReward = block.timestamp.sub(pool.lastRewardTimestamp).mul(pool.amountPerSecond);
230:            if (block.timestamp > pool.endDate) {
231:                currentReward = pool.endDate.sub(pool.lastRewardTimestamp).mul(pool.amountPerSecond);
232:                pool.lastRewardTimestamp = pool.endDate;
233:            } else {
234:                pool.lastRewardTimestamp = block.timestamp;
235:            }
236:            pool.accAmountPerShare = pool.accAmountPerShare.add(currentReward.mul(1e12).div(pool.totalLP));
237:            pool.totalReward = pool.totalReward.add(currentReward);
238:        }
239:    }
```

Description

biakia : In contract `EchodexFarm`, there is a possibility of over-distribution of rewards, and here is an example: We assume that `startDate` is 10000, `endDate` is 20000 and `amountPerSecond` is 1. So the total rewards will be `1 * (20000-10000) = 10000`.

1. Alice calls `stake` to stake 1 LP token at time 10000, The `pool.accAmountPerShare` will be 0. The `pool.lastRewardTimestamp` will be updated to 10000 in function `stake` because `pool.lastRewardTimestamp` now is 0:

```
if (pool.lastRewardTimestamp == 0) {
            pool.lastRewardTimestamp = block.timestamp;
}
```

2. Alice calls `unstake` to withdraw 1 LP token at time 15000. The `pool.accAmountPerShare` will be `(15000-10000)*1*1e12/1 = 5000e12` and `pool.lastRewardTimestamp` will be 15000.

```
if (pool.totalLP > 0 && pool.lastRewardTimestamp <= pool.endDate) {
            uint256 currentReward = block.timestamp.sub(pool.lastRewardTimestamp).mul(pool.amountPer
Second);
            if (block.timestamp > pool.endDate) {
                currentReward = pool.endDate.sub(pool.lastRewardTimestamp).mul(pool.amountPerSecon
d);
                pool.lastRewardTimestamp = pool.endDate;
            } else {
                pool.lastRewardTimestamp = block.timestamp;
            }
            pool.accAmountPerShare = pool.accAmountPerShare.add(currentReward.mul(1e12).div(pool.tot
alLP));
            pool.totalReward = pool.totalReward.add(currentReward);
        }
```

The `user.rewardEarn` will be `1 * 5000e12 / 1e12 - 0 = 5000`:

```
function _audit(User storage user, Pool storage pool) private {
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accAmountPerShare).div(1e12).sub(user.rewardDeb
t);
            user.rewardEarn = user.rewardEarn.add(pending);
            user.rewardDebt = user.amount.mul(pool.accAmountPerShare).div(1e12);
        }
    }
```

3. Bob calls `stake` to stake 1 LP token at 16000. At this time, the `pool.accAmountPerShare` will not be updated because `pool.totalLP` is 0 so `pool.accAmountPerShare` is still `5000e12` now. The `pool.lastRewardTimestamp` will not be updated too, so the `pool.lastRewardTimestamp` is still 15000:
4. Bob calls `unstake` to withdraw 1 LP token at time 20000. At this time, the `pool.accAmountPerShare` will be `5000e12 + (20000-15000)*1*1e12/1 = 10000e12`, the `user.rewardEarn` will be `1 * 10000e12 / 1e12 - 5000 = 5000`.
5. During 15000~16000, there is no user staking in the contract, so there are some excess rewards. These rewards are recorded in `pool.totalExcessReward`. When Alices calls `unstake` at 15000, the `pool.startTimeExcess` will be updated to 15000.

```
if (pool.totalLP == 0) {
            pool.startTimeExcess = block.timestamp;
 }
```

When Bob calls `stake` at 16000, the `pool.totalExcessReward` will be updated to `1000(16000-15000)`.

```
if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(block.timestamp.sub(pool.startTimeEx
cess));
            pool.startTimeExcess = 0;
}
```

These rewards can be withdraw by the pool owner:

```
function withdrawExcessReward(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.owner == msg.sender, "EchodexFarm: NO_PERMISSION");
        require(pool.endDate < block.timestamp, "EchodexFarm: POOL_NOT_END");

        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(pool.endDate.sub(pool.startTimeExces
s));
            pool.startTimeExcess = 0;
        }

        require(pool.totalExcessReward > 0, "EchodexFarm: NO_EXCESS");

        _safeTransfer(pool.tokenReward, msg.sender, pool.totalExcessReward.mul(pool.amountPerSecon
d));
        emit WithdrawExcess(poolId, pool.totalExcessReward);
        pool.totalExcessReward = 0;
    }
```

At last, Alice can get 5000 tokens, Bob can get 5000 tokens and the owner can get back 1000 tokens. The total tokens is `5000 + 5000 + 1000 = 11000`, which is greater than the total rewards.

If the owner calls `withdrawExcessReward` first to retrieve 1000 tokens, then Alice or Bob may fail to call `harvest` because there are not enough tokens in the contract.

Recommendation

biakia : Consider updating `pool.lastRewardTimestamp` to the current time even when `pool.totalLP` is equal to 0:

```
function _update(Pool storage pool) private {
        if(pool.totalLP == 0){
           pool.lastRewardTimestamp = block.timestamp;
        }
        if (pool.totalLP > 0 && pool.lastRewardTimestamp <= pool.endDate) {
            uint256 currentReward = block.timestamp.sub(pool.lastRewardTimestamp).mul(pool.amountPer
Second);
            if (block.timestamp > pool.endDate) {
                currentReward = pool.endDate.sub(pool.lastRewardTimestamp).mul(pool.amountPerSecon
d);
                pool.lastRewardTimestamp = pool.endDate;
            } else {
                pool.lastRewardTimestamp = block.timestamp;
            }
            pool.accAmountPerShare = pool.accAmountPerShare.add(currentReward.mul(1e12).div(pool.tot
alLP));
            pool.totalReward = pool.totalReward.add(currentReward);
        }
    }
```

Client Response

Fixed,Fixed wrong `lastRewardTimestamp` cause reward over-distribution

ECD-3: `amountsFeeAddMore` processing logic errors, which may cause asset loss or swap failure

Category	Severity	Status	Contributor
Logical	Critical	Mitigated	Xi_Zi

Code Reference

- code/contracts/EchodexRouterFee.sol#L256-L263

```
256:            if (amountsFeeAddMore[i] > 0) {
257:                address tokenFee = IEchodexFactory(factory).tokenFee();
258:                if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
259:                    IERC20(tokenFee).approve(pair, uint256(-1));
260:                }
261:                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
262:                IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
263:            }
```

Description

Xi_Zi : In the EchodexRouterFee contract, amountsFeeAddMore is passed in for fee charge when performing swap-related function operations. However, amountsFeeAddMore is processed by _swap function. Only the value of amountsFeeAddMore is greater than zero for processing, and the processing does not call `IEchodexFactory(factory).calcFee(amountOut, tokenOut, address(this), factory) in advance;` For the calculation of fee, if the value of amountsFeeAddMore passed in by the user is greater than the actual fee, there is no relevant refund logic, which may cause damage to the user's funds。 In addition, it is necessary to determine whether the amountsFeeAddMore passed in by the user is greater than or equal to the value of factory.calcFee. If the user fee is insufficient, an error will occur in swap due to insufficient k-value check

```
        function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline,
        uint[] calldata amountsFeeAddMore
    ) external virtual ensure(deadline) returns (uint256[] memory amounts) {
        amounts = EchodexLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, "EchodexRouter: EXCESSIVE_INPUT_AMOUNT");
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            EchodexLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, to, amountsFeeAddMore);
    }


    function _swap(
        uint256[] memory amounts,
        address[] memory path,
        address _to,
        uint[] memory amountsFeeAddMore//@audit 3.0 calldata
    ) internal virtual {
        for (uint256 i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = EchodexLibrary.sortTokens(input, output);
            uint256 amountOut = amounts[i + 1];
            (uint256 amount0Out, uint256 amount1Out) =
                input == token0 ? (uint256(0), amountOut) : (amountOut, uint256(0));
            address to = i < path.length - 2 ? EchodexLibrary.pairFor(factory, output, path[i + 2])
: _to;
            address pair = EchodexLibrary.pairFor(factory, input, output);
            if (amountsFeeAddMore[i] > 0) {
                address tokenFee = IEchodexFactory(factory).tokenFee();
                if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
                    IERC20(tokenFee).approve(pair, uint256(-1));
                }
                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);/
                IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
```

```
  }
              IEchodexPair(pair).swapPayWithTokenFee(amount0Out, amount1Out, to, msg.sender, new bytes
  (0));
          }
      }
```

Recommendation

Xi_Zi : Add the amountsFeeAddMore checksum

```
    function _swap(
          uint256[] memory amounts,
          address[] memory path,
          address _to,
          uint[] memory amountsFeeAddMore//@audit 3.0 calldata
      ) internal virtual {
          for (uint256 i; i < path.length - 1; i++) {
              (address input, address output) = (path[i], path[i + 1]);
              (address token0,) = EchodexLibrary.sortTokens(input, output);
              uint256 amountOut = amounts[i + 1];
              (uint256 amount0Out, uint256 amount1Out) =
                  input == token0 ? (uint256(0), amountOut) : (amountOut, uint256(0));
              address to = i < path.length - 2 ? EchodexLibrary.pairFor(factory, output, path[i + 2])
  : _to;
              address pair = EchodexLibrary.pairFor(factory, input, output);
              (uint fee,)=IEchodexFactory(factory).calcFee(amountOut, output, address(this), factory);
              require(amountsFeeAddMore[i] >= fee);
              address tokenFee = IEchodexFactory(factory).tokenFee();
              if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
                  IERC20(tokenFee).approve(pair, uint256(-1));
              }
              IERC20(tokenFee).transferFrom(msg.sender, address(this), fee);
              IEchodexPair(pair).addFee(fee);

              IEchodexPair(pair).swapPayWithTokenFee(amount0Out, amount1Out, to, msg.sender, new bytes
  (0));

          }
      }
```

Client Response

Mitigated,We will simulate exact `amountsFeeAddMore` on interface to avoid asset loss and swap failure

ECD-4:old `tokenFee` will be locked in the pair after `tokenFee` is updated

Category	Severity	Status	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/contracts/EchodexPair.sol#L230-L237

```
230:    function addFee(uint amount) external lock {
231:        address tokenFee = IEchodexFactory(factory).tokenFee();
232:        IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);
233:        totalFee = totalFee + amount;
234:        currentFee = currentFee + amount;
235:
236:        emit AddFee(amount);
237:    }
```

Description

biakia : In contract `EchodexPair`, the function `addFee` will add `tokenFee` to the pair:

```
function addFee(uint amount) external lock {
        address tokenFee = IEchodexFactory(factory).tokenFee();
        IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);
        totalFee = totalFee + amount;
        currentFee = currentFee + amount;

        emit AddFee(amount);
    }
```

It will call `transferFrom` first and then add `amount` to the `currentFee`. In contract `EchodexFactory`, the owner can call `setTokenFee` to change the `tokenFee`:

```
function setTokenFee(address _tokenFee) external {
        require(msg.sender == owner, 'Echodex: FORBIDDEN');
        tokenFee = _tokenFee;
}
```

Let's say the `tokenFee` is `WETH` now and a pair has 1e18 WETH as fee, so the `currentFee` is 1e18 now. Then the owner calls `setTokenFee` to change the `tokenFee` from `WETH` to `USDC`. At this time, the `currentFee` is still 1e18 but the pair doesn't have 1e18 `USDC`. When someone calls `swapPayWithTokenFee`, the function `_payFee` will revert because there is not enough `USDC` in the pair. What's more, there is no function to withdraw those `WETH` in the pair. All these `WETH` will be locked in the pair forever.

Recommendation

biakia : Consider reseting `currentFee` and withdrawing all old `tokenFee` if `tokenFee` is updated.

Client Response

Fixed, Added withdrawFee function

ECD-5:Possible failure of the call `withdrawExcessReward` function

Category	Severity	Status	Contributor
Logical	**Medium**	**Fixed**	**biakia**

Code Reference

- code/contracts/EchodexFarm.sol#L205-L220
- code/contracts/EchodexFarm.sol#L162-L185

```
    function unstake(uint256 poolId, uint256 amountLP) external {
        require(amountLP > 0 , "EchodexFarm: AMOUNT_LP_NOT_ZERO");

        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];
        require(amountLP <= user.amount , "EchodexFarm: INSUFFICIENT_AMOUNT");

        _update(pool);
        _audit(user, pool);

        _safeTransfer(pool.pairAddress, msg.sender, amountLP);

        pool.totalLP = pool.totalLP.sub(amountLP);
        user.amount = user.amount.sub(amountLP);
        user.rewardDebt = user.amount.mul(pool.accAmountPerShare).div(1e12);

        if (pool.totalLP == 0) {
            pool.startTimeExcess = block.timestamp;
        }

        emit PoolUpdate(poolId, pool.accAmountPerShare, pool.totalLP, pool.totalReward, pool.lastRewardTimestamp);
        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
        emit Unstake(poolId, msg.sender, amountLP);
    }

    function withdrawExcessReward(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.owner == msg.sender, "EchodexFarm: NO_PERMISSION");
        require(pool.endDate < block.timestamp, "EchodexFarm: POOL_NOT_END");

        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(pool.endDate.sub(pool.startTimeExcess));
            pool.startTimeExcess = 0;
        }

        require(pool.totalExcessReward > 0, "EchodexFarm: NO_EXCESS");

        _safeTransfer(pool.tokenReward, msg.sender, pool.totalExcessReward.mul(pool.amountPerSecond));
```

```
218:        emit WithdrawExcess(poolId, pool.totalExcessReward);
219:        pool.totalExcessReward = 0;
220:    }
```

Description

biakia : In contract `EchodexFarm`, the function `withdrawExcessReward` is used to retrieve undistributed rewards:

```
function withdrawExcessReward(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.owner == msg.sender, "EchodexFarm: NO_PERMISSION");
        require(pool.endDate < block.timestamp, "EchodexFarm: POOL_NOT_END");

        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(pool.endDate.sub(pool.startTimeExces
s));
            pool.startTimeExcess = 0;
        }

        require(pool.totalExcessReward > 0, "EchodexFarm: NO_EXCESS");

        _safeTransfer(pool.tokenReward, msg.sender, pool.totalExcessReward.mul(pool.amountPerSecon
d));
        emit WithdrawExcess(poolId, pool.totalExcessReward);
        pool.totalExcessReward = 0;
    }
```

It will calculate rewards based on `pool.startTimeExcess` and `pool.endDate`:

```
        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(pool.endDate.sub(pool.startTimeExces
s));
            pool.startTimeExcess = 0;
        }
```

The `pool.endDate` is initialized in function `createPool` and will never be changed. The `pool.startTimeExcess` will be updated in function `unstake`:

```
function unstake(uint256 poolId, uint256 amountLP) external {
        require(amountLP > 0 , "EchodexFarm: AMOUNT_LP_NOT_ZERO");

        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];
        require(amountLP <= user.amount , "EchodexFarm: INSUFFICIENT_AMOUNT");

        _update(pool);
        _audit(user, pool);

        _safeTransfer(pool.pairAddress, msg.sender, amountLP);

        pool.totalLP = pool.totalLP.sub(amountLP);
        user.amount = user.amount.sub(amountLP);
        user.rewardDebt = user.amount.mul(pool.accAmountPerShare).div(1e12);

        if (pool.totalLP == 0) {
            pool.startTimeExcess = block.timestamp;
        }

        emit PoolUpdate(poolId, pool.accAmountPerShare, pool.totalLP, pool.totalReward, pool.lastRew
ardTimestamp);
        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
        emit Unstake(poolId, msg.sender, amountLP);
    }
```

It is possible that the `block.timestamp` is greater than `pool.endDate` when the user calls function `unstake`. As a result, the `pool.startTimeExcess` will be greater than `pool.endDate` and the calculation `pool.endDate.sub(pool.startTimeExcess)` in function `withdrawExcessReward` will fail.

Recommendation

biakia : Consider making sure `pool.startTimeExcess` is not greater than `pool.endDate` in function `unstake`:

```
if (pool.totalLP == 0) {
            pool.startTimeExcess = block.timestamp;
            if(pool.startTimeExcess>pool.endDate){
                 pool.startTimeExcess=pool.endDate;
            }
}
```

Client Response

Fixed, Fixed wrong `lastRewardTimestamp` cause reward over-distribution

ECD-6:Incorrect input parameters for flash swap

Category	Severity	Status	Contributor
Logical	**Medium**	**Fixed**	**biakia**

Code Reference

- code/contracts/EchodexPair.sol#L180-L203

```
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock { // payWithTokenFee = false
        SwapState memory state = _preSwap(amount0Out, amount1Out, to);

        uint amountOut = amount0Out > 0 ? amount0Out : amount1Out;
        address tokenOut = amount0Out > 0 ? token0 : token1;
        uint fee = amountOut.mul(3) / 1000;
        amountOut = amountOut.sub(fee);
        _safeTransfer(tokenOut, to, amountOut);
        _safeTransfer(tokenOut, IEchodexFactory(factory).receiveFeeAddress(), fee);

        if (data.length > 0) IEchodexCallee(to).echodexCall(msg.sender, amount0Out, amount1Out, data);
        state.balance0 = IERC20(token0).balanceOf(address(this));
        state.balance1 = IERC20(token1).balanceOf(address(this));

        state.amount0In = state.balance0 > state._reserve0 - amount0Out ? state.balance0 - (state._reserve0 - amount0Out) : 0;
        state.amount1In = state.balance1 > state._reserve1 - amount1Out ? state.balance1 - (state._reserve1 - amount1Out) : 0;
        require(state.amount0In > 0 || state.amount1In > 0, 'Echodex: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reserve1), 'Echodex: K');
        }

        _update(state.balance0, state.balance1, state._reserve0, state._reserve1);
        emit Swap(msg.sender, state.amount0In, state.amount1In, amount0Out, amount1Out, to, 0 , 0);
    }
```

Description

biakia : The function `swap` supports flash swap only when the input param `data` is not empty:

```
if (data.length > 0) IEchodexCallee(to).echodexCall(msg.sender, amount0Out, amount1Out, data);
```

The `to` address should implement the following function:

```
  function echodexCall(
          address sender,
          uint256 amount0,
          uint256 amount1,
          bytes calldata data
      ) external;
```

The `amount0` or `amount1` is the actual amount of the token you will receive. In the function `swap` it will charge a 0.3% fee directly:

```
          uint amountOut = amount0Out > 0 ? amount0Out : amount1Out;
          address tokenOut = amount0Out > 0 ? token0 : token1;
          uint fee = amountOut.mul(3) / 1000;
          amountOut = amountOut.sub(fee);
          _safeTransfer(tokenOut, to, amountOut);
          _safeTransfer(tokenOut, IEchodexFactory(factory).receiveFeeAddress(), fee);
```

so the actual amount of token you receive is `amountOut` instead of `amount0Out` or `amount1Out`. However, `amount0Out` and `amount1Out` are still used when calling the function `echodexCall`.

Recommendation

biakia : Consider using the `amountOut` when calling the function `echodexCall`:

```
   if (data.length > 0){
      if(amount0Out>0){
          IEchodexCallee(to).echodexCall(msg.sender, amountOut, amount1Out, data);
      }else if(amount1Out>0){
          IEchodexCallee(to).echodexCall(msg.sender, amount0Out, amountOut, data);
      }
  }
```

Client Response

Fixed, Flash swap now available in `swap` function and `swapPayWithTokenFee` function

ECD-7:Use `safeTransfer` instead of `transfer`

Category	Severity	Status	Contributor
Logical	**Medium**	**Fixed**	**ginlee, ladboy233**

Code Reference

- code/contracts/EchodexPair.sol#L16
- code/contracts/EchodexPair.sol#L56-L59
- code/contracts/EchodexRouterFee.sol#L259
- code/contracts/EchodexRouterFee.sol#L261

```
16:    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

56:    function _safeTransfer(address token, address to, uint value) private {
57:        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
58:        require(success && (data.length == 0 || abi.decode(data, (bool))), 'Echodex: TRANSFER_FAILED');
59:    }

259:                    IERC20(tokenFee).approve(pair, uint256(-1));

261:                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
```

Description

ginlee : _safeTransfer does not uses SafeERC20 library as the function name implies

```
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));
```

SELECTOR is hashed with transfer function, then in _safeTransfer it is used for token transfer

```
   (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
```

The ERC20.transfer() and ERC20.transferFrom() functions return a boolean value indicating success. This parameter needs to be checked for success. Some tokens do not revert if the transfer failed but return false instead.

ladboy233 : Across the codebase, the code use transfer and transferFrom and approve

However,

https://github.com/d-xo/weird-erc20#missing-return-values

Some tokens do not return a bool (e.g. USDT, BNB, OMG) on ERC20 methods. see here for a comprehensive (if somewhat outdated) list.

Some tokens (e.g. BNB) may return a bool for some methods, but fail to do so for others. This resulted in stuck BNB tokens in Uniswap v1 (details).

Some particularly pathological tokens (e.g. Tether Gold) declare a bool return, but then return false even when the transfer was successful (code).

A good safe transfer abstraction (example) can help somewhat, but note that the existence of Tether Gold makes it impossible to correctly handle return values for all tokens.

for token such as USDT, the token does not return a boolean in approve function so call IERC20(tokenFee).approve will result in revert

the protocol should use safeTransfer and safeApprove to support a wide range of ERC20 instead the non-standarded but widely adopted ERC20 token

Recommendation

ginlee : Recommend using OpenZeppelin's SafeERC20 versions with the safeTransfer functions that handle the return value check as well as non-standard-compliant tokens.

ladboy233 : use safeTransfer and safeApprove

Client Response

Fixed

ECD-8:Potential reentrancy risk in `EchodexFarm::harvest()`

Category	Severity	Status	Contributor
Reentrancy	**Medium**	**Fixed**	**biakia, ginlee, ladboy233**

Code Reference

- code/contracts/EchodexFarm.sol#L187-L203
- code/contracts/EchodexFarm.sol#L205-L220
- code/contracts/EchodexFarm.sol#L162-L185
- code/contracts/EchodexFarm.sol#L131-L160
- code/contracts/EchodexFarm.sol#L99-L129
- code/contracts/EchodexFarm.sol#L187-L232

```
    function createPool(address tokenA, address tokenB, uint256 amountReward, address tokenReward, uint256 startDate, uint256 endDate) external {
        require(block.timestamp <= startDate, "EchodexFarm: WRONG_TIME");
        require(startDate + 30 * 60 <= endDate, "EchodexFarm: WRONG_TIME");

        address pairAddress = IEchodexFactory(factory).getPair(tokenA, tokenB);
        require(pairAddress != address(0), "EchodexFarm: PAIR_NOT_EXIST");


        uint256 amountPerSecond = amountReward.div(endDate.sub(startDate));
        pools[currentPoolId] = Pool({
            poolId: currentPoolId,
            pairAddress: pairAddress,
            amountReward: amountReward,
            tokenReward: tokenReward,
            startDate: startDate,
            endDate: endDate,
            accAmountPerShare: 0,
            totalLP: 0,
            totalReward: 0,
            amountPerSecond: amountPerSecond,
            lastRewardTimestamp: 0,
            owner: msg.sender,
            startTimeExcess: startDate,
            totalExcessReward: 0
        });

        TransferHelper.safeTransferFrom(tokenReward, msg.sender, address(this), amountReward);
        emit PoolCreated(currentPoolId, pairAddress, tokenA, tokenB, amountReward, tokenReward, startDate, endDate, amountPerSecond);

        currentPoolId++;
    }

    function stake(uint256 poolId, uint256 amountLP) external {
        require(amountLP > 0 , "EchodexFarm: AMOUNT_LP_NOT_ZERO");

        Pool storage pool = pools[poolId];
        require(pool.startDate <= block.timestamp, "EchodexFarm: NOT_START");
        require(block.timestamp <= pool.endDate, "EchodexFarm: OVER_TIME");

        if (pool.lastRewardTimestamp == 0) {
            pool.lastRewardTimestamp = block.timestamp;
        }

        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(block.timestamp.sub(pool.startTimeExcess));
            pool.startTimeExcess = 0;
        }

        User storage user = users[msg.sender][poolId];

        _update(pool);
        _audit(user, pool);

        TransferHelper.safeTransferFrom(pool.pairAddress, msg.sender, address(this), amountLP);
        pool.totalLP = pool.totalLP.add(amountLP);
        user.amount = user.amount.add(amountLP);
        user.rewardDebt = user.amount.mul(pool.accAmountPerShare).div(1e12);

        emit PoolUpdate(poolId, pool.accAmountPerShare, pool.totalLP, pool.totalReward, pool.lastRewardTimestamp);
        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
        emit Stake(poolId, msg.sender, amountLP);
    }

    function unstake(uint256 poolId, uint256 amountLP) external {
        require(amountLP > 0 , "EchodexFarm: AMOUNT_LP_NOT_ZERO");

        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];
        require(amountLP <= user.amount , "EchodexFarm: INSUFFICIENT_AMOUNT");

        _update(pool);
        _audit(user, pool);

        _safeTransfer(pool.pairAddress, msg.sender, amountLP);

        pool.totalLP = pool.totalLP.sub(amountLP);
        user.amount = user.amount.sub(amountLP);
        user.rewardDebt = user.amount.mul(pool.accAmountPerShare).div(1e12);

        if (pool.totalLP == 0) {
            pool.startTimeExcess = block.timestamp;
        }

        emit PoolUpdate(poolId, pool.accAmountPerShare, pool.totalLP, pool.totalReward, pool.lastRewardTimestamp);
        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
        emit Unstake(poolId, msg.sender, amountLP);
    }

    function harvest(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];

        _update(pool);
        _audit(user, pool);

        require(user.rewardEarn > 0, "EchodexFarm: NO_REWARD");

        _safeTransfer(pool.tokenReward, msg.sender, user.rewardEarn);

        emit Harvest(poolId, msg.sender, user.rewardEarn);

        user.rewardEarn = 0;

        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
    }

    function harvest(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];

        _update(pool);
        _audit(user, pool);

        require(user.rewardEarn > 0, "EchodexFarm: NO_REWARD");

        _safeTransfer(pool.tokenReward, msg.sender, user.rewardEarn);

        emit Harvest(poolId, msg.sender, user.rewardEarn);

        user.rewardEarn = 0;

        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
    }

    function withdrawExcessReward(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.owner == msg.sender, "EchodexFarm: NO_PERMISSION");
        require(pool.endDate < block.timestamp, "EchodexFarm: POOL_NOT_END");

        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(pool.endDate.sub(pool.startTimeExcess));
            pool.startTimeExcess = 0;
        }

        require(pool.totalExcessReward > 0, "EchodexFarm: NO_EXCESS");

        _safeTransfer(pool.tokenReward, msg.sender, pool.totalExcessReward.mul(pool.amountPerSecond));
        emit WithdrawExcess(poolId, pool.totalExcessReward);
        pool.totalExcessReward = 0;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'EchodexFarm: TRANSFER_FAILED');
    }

    function _update(Pool storage pool) private {
        if (pool.totalLP > 0 && pool.lastRewardTimestamp <= pool.endDate) {
            uint256 currentReward = block.timestamp.sub(pool.lastRewardTimestamp).mul(pool.amountPerSecond);
            if (block.timestamp > pool.endDate) {
                currentReward = pool.endDate.sub(pool.lastRewardTimestamp).mul(pool.amountPerSecond);
                pool.lastRewardTimestamp = pool.endDate;

    function withdrawExcessReward(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.owner == msg.sender, "EchodexFarm: NO_PERMISSION");
        require(pool.endDate < block.timestamp, "EchodexFarm: POOL_NOT_END");

        if (pool.startTimeExcess != 0) {
            pool.totalExcessReward = pool.totalExcessReward.add(pool.endDate.sub(pool.startTimeExcess));
            pool.startTimeExcess = 0;
        }

        require(pool.totalExcessReward > 0, "EchodexFarm: NO_EXCESS");

        _safeTransfer(pool.tokenReward, msg.sender, pool.totalExcessReward.mul(pool.amountPerSecond));
        emit WithdrawExcess(poolId, pool.totalExcessReward);
        pool.totalExcessReward = 0;
    }
```

Description

biakia : A reentrancy attack can occur when the contract creates a function that makes an external call to another contract before resolving any effects. In function `harvest`, it will transfer tokens first and then update the `user.rewardEarn` to 0:

```
function harvest(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];

        _update(pool);
        _audit(user, pool);

        require(user.rewardEarn > 0, "EchodexFarm: NO_REWARD");

        _safeTransfer(pool.tokenReward, msg.sender, user.rewardEarn);

        emit Harvest(poolId, msg.sender, user.rewardEarn);

        user.rewardEarn = 0;

        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
    }
```

If the `pool.tokenReward` is a ERC777 token, it is possible for anyone to perform a reentrancy attack. As a result, all rewards will be stolen from the pool.

The same issue exits in the `withdrawExcessReward` function.

ginlee :

```
    function harvest(uint256 poolId) external {
            Pool storage pool = pools[poolId];
            User storage user = users[msg.sender][poolId];

            _update(pool);
            _audit(user, pool);

            require(user.rewardEarn > 0, "EchodexFarm: NO_REWARD");

            _safeTransfer(pool.tokenReward, msg.sender, user.rewardEarn);

            emit Harvest(poolId, msg.sender, user.rewardEarn);

            user.rewardEarn = 0;

            emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
      }
```

set user.rewardEarn after _safeTransfer, state changes after interaction, which is an obvious reentrancy exploit design pattern same issue in createPool function, stake function and unstake function in this contract

ladboy233 : the ExchodexFarm is meanted to support a LP token with any reward token to give user incentive to stake and lock their LP token

If the reward token in EchoFarming is ERC777 and support callback, the same token ERC777 token in the farming contract can be drained

In function unstake, harvest and withdrawExcessReward

the code violates the check-effect-pattern and update state after the external transfer

In function unstake, this is not a problem because we know the LP token does not support callback

```
  _safeTransfer(pool.pairAddress, msg.sender, amountLP);

  pool.totalLP = pool.totalLP.sub(amountLP);
  user.amount = user.amount.sub(amountLP);
  user.rewardDebt = user.amount.mul(pool.accAmountPerShare).div(1e12);
```

However, in harvest function and withdrawExcessReward,

this can be issue if the reward token is a ERC777 token with callback

https://eips.ethereum.org/EIPS/eip-777#hooks

the same token can be drained by re-enter the harvest or withdrawExcessReward because the reward state is cleared up after the exteral transfer

```
    function harvest(uint256 poolId) external {
        Pool storage pool = pools[poolId];
        User storage user = users[msg.sender][poolId];

        _update(pool);
        _audit(user, pool);

        require(user.rewardEarn > 0, "EchodexFarm: NO_REWARD");

        // @audit
        // reentrancy
        _safeTransfer(pool.tokenReward, msg.sender, user.rewardEarn);

        emit Harvest(poolId, msg.sender, user.rewardEarn);

        user.rewardEarn = 0;

        emit UserUpdate(msg.sender, poolId, user.amount, user.rewardDebt, user.rewardEarn);
    }
```

Past attack that leverage the ERC777 callback:

https://milkroad.com/news/imbtc-uniswap-hack/

Recommendation

biakia : We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the function `harvest`.
ginlee : Use the Checks-Effects-Interactions and make all state changes before calling external contracts. Consider using function modifiers such as nonReentrant from Openzeppelin ReentrancyGuard library to prevent re-entrancy.
ladboy233 : Add reentrancy guard in harvest and withdrawExcessReward function and update the state before the external transfer

Client Response

Fixed

ECD-9:Missing `nonce` and `chain_id` in `EchodexRouterFee` contract, `removeLiquidityWithPermit` function may lead to signature replay attacks

Category	Severity	Status	Contributor
Signature Forgery or Replay	**Medium**	**Acknowledged**	**ginlee**

Code Reference

- code/contracts/EchodexRouterFee.sol#L163-L180

```
163:    function removeLiquidityWithPermit(
164:        address tokenA,
165:        address tokenB,
166:        uint256 liquidity,
167:        uint256 amountAMin,
168:        uint256 amountBMin,
169:        address to,
170:        uint256 deadline,
171:        bool approveMax,
172:        uint8 v,
173:        bytes32 r,
174:        bytes32 s
175:    ) external virtual returns (uint256 amountA, uint256 amountB) {
176:        address pair = EchodexLibrary.pairFor(factory, tokenA, tokenB);
177:        uint256 value = approveMax ? uint256(-1) : liquidity;
178:        IEchodexPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
179:        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin,
to, deadline);
180:    }
```

Description

ginlee : In removeLiquidityWithPermit function, params have no nonce and chain_id, which may lead to signature replay attacks

1. To prevent signature replay attacks, smart contracts must:

- keep track of a nonce
- make the current nonce available to signers,
- validate the signature using the current nonce,
- once a nonce has been used, save this to storage such that the same nonce can't be used again.

2.Many smart contracts operate on multiple chains from the same contract address and users similarly operate the same address across multiple chains a valid signature that was used on one chain could be copied by an attacker and propagated onto another chain, where it would also be valid for the same user & contract address

Recommendation

ginlee : 1.This requires signers to sign their message including the current nonce, and hence signatures that have already been used are unable to be replayed, as the old nonce will have been marked in storage as having been used & will no longer be valid. An example can be seen in OpenZeppelin's ERC20Permit (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Permit.sol#L60-L93)implementation

2.To prevent cross-chain signature replay attacks, smart contracts must validate the signature using the chain_id, and users must include the chain_id in the message to be signed

Client Response

Acknowledged, `nonce` and `chain_id` will be validate in `EchodexPair.permit()`

ECD-10:Need input validation for `stake` function in `EchodexFarm` contract

Category	Severity	Status	Contributor
Logical	Medium	Fixed	ginlee

Code Reference

- code/contracts/EchodexFarm.sol#L131-L134

```
131:    function stake(uint256 poolId, uint256 amountLP) external {
132:        require(amountLP > 0 , "EchodexFarm: AMOUNT_LP_NOT_ZERO");
133:
134:        Pool storage pool = pools[poolId];
```

Description

ginlee : In Solidity, developers often assume that reading a non-existent index in a mapping will cause the Solidity program to revert (i.e., interrupt execution and undo all previous changes). However, in reality, when developers attempt to read a non-existent index, Solidity does not revert; instead, it returns an empty object with default member values.

```
Pool storage pool = pools[poolId]
```

If poolId doesn't exist, the mapping will return a Pool object with default values

Recommendation

ginlee : If developers need to look up a value in a mapping, they should first check if the value exists in the mapping before reading it to avoid unnecessary errors or security issues. For example, one can determine the existence of a value by checking if the specified key exists in the mapping.

Client Response

Fixed

ECD-11:Use spot price as fee amount oracle is vulnerable for manipulation

Category	Severity	Status	Contributor
Oracle Manipulation	**Medium**	**Mitigated**	**ladboy233**

Code Reference

- code/contracts/EchodexRouterFee.sol#L264
- code/contracts/EchodexPair.sol#L213
- code/contracts/EchodexPair.sol#L109
- code/contracts/EchodexFactory.sol#L82

```
82:        uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);

109:        require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');

213:        _payFee(fee, feeRefund, refundFeeAddress);

264:            IEchodexPair(pair).swapPayWithTokenFee(amount0Out, amount1Out, to, msg.sender, new bytes(0));
```

Description

ladboy233 : In the code EchodexRouterFee.sol,
In swap function,
we need to pay attention to this newly added code:

```
if (amountsFeeAddMore[i] > 0) {
                address tokenFee = IEchodexFactory(factory).tokenFee();
                if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
                        IERC20(tokenFee).approve(pair, uint256(-1));
                }
                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
                IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
        }
        IEchodexPair(pair).swapPayWithTokenFee(amount0Out, amount1Out, to, msg.sender, new bytes
(0));
```

this is calling swapPayWithTokenFee on the EchodexPair contract

We are calling:

```
(uint fee, uint feeRefund) = IEchodexFactory(factory).calcFee(amountOut, tokenOut, address(this), fa
ctory);
_payFee(fee, feeRefund, refundFeeAddress);
_safeTransfer(tokenOut, to, amountOut);
```

this first calling IEchodexFactory(factory).calcFee, which calls:

```
    function setFeePath(address tokenOut, address[] calldata path) external {
        require(msg.sender == owner, 'Echodex: FORBIDDEN');
        feePath[tokenOut] = path;
        feePathLength[tokenOut] = path.length;
    }

    // @audit
    function calcFee(uint amountOut, address tokenOut, address pair, address factory) external view
returns (uint fee, uint feeRefund) {
        uint amountFeeTokenOut = amountOut / 1000;

        uint amountFeeRefundTokenOut = 0;
        feeRefund = 0;
        if (refundPercent[pair] > 0) {
            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund
(0.05 * 10 **18)% fee
        }

        address[] memory path = feePath[tokenOut];
        uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
        fee = amounts[amounts.length - 1];

        if (amountFeeRefundTokenOut > 0) {
            uint256[] memory amountsRefund = EchodexLibrary.getAmountsOut(factory, amountFeeRefundTo
kenOut, path);
            feeRefund = amountsRefund[amountsRefund.length - 1];
        }
    }
```

this is very problematic

when calcuting the fee, we are using spot price oracle to calculate the fee amount to the pay the protocol receiver and the refund amount,

```
uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
```

the path is set by admin

because we are using the spot price,

another malicious user can manipulate the liquidity of the pool to make the returned fee amount too small and the fee amount and the refund amount will not be transferred to the receiver and the refund address

ok, we need to make some assumption:

the pool the user is trade is ETH / USDC

the fee token is USDC and the pool that determine the amount of fee token paid to protocol and refund address is the pool XYZ / USDC, I am referring to the feePath set by admin

```
function setFeePath(address tokenOut, address[] calldata path) external {
        require(msg.sender == owner, 'Echodex: FORBIDDEN');
        feePath[tokenOut] = path;
        feePathLength[tokenOut] = path.length;
}
```

an attacker monitor the mempool and see the victim transaction.

1. an attacker make a trade on the pool XYZ / USDC make the USDC fee amount returned very low and close to 0
2. user's trade is done in pool ETH / USDC but the fee token is in USDC, since the code use spot price oracle to determine the fee amount, the attacker already manipulate the liquidity of the pool XYZ / USDC, the fee amount returned is too low
3. the protocol receiver and refund address will not receive fee
4. because the user's trading pool is ETH / USDC and the fee token is USDC, the USDC fee remains in the pool and inflate the ETH price
5. an attacker can make an opposite trade to recover the liquidity of the pool XYZ / USDC
6. an attacker can sell ETH at higher price and steal the fee token USDC that are remaining in the ETH / USDC pool

such attack is feasible because:

1. the code use the spot price oracle to determine the fee amount
2. if the fee token is the same as either token0 or token1 of the liquidity pool the user is trading (if this case, the user is trading on the pool ETH / USDC and the fee token is USDC)
   ladboy233 : In the code EchodexRouterFee.sol,

In swap function,

we need to pay attention to this newly added code:

```
if (amountsFeeAddMore[i] > 0) {
                address tokenFee = IEchodexFactory(factory).tokenFee();
                if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
                        IERC20(tokenFee).approve(pair, uint256(-1));
                }
                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
                IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
        }
        IEchodexPair(pair).swapPayWithTokenFee(amount0Out, amount1Out, to, msg.sender, new bytes
(0));
```

this is calling swapPayWithTokenFee on the EchodexPair contract

We are calling:

```
(uint fee, uint feeRefund) = IEchodexFactory(factory).calcFee(amountOut, tokenOut, address(this), fa
ctory);
_payFee(fee, feeRefund, refundFeeAddress);
_safeTransfer(tokenOut, to, amountOut);
```

this first calling IEchodexFactory(factory).calcFee, which calls:

```

    function setFeePath(address tokenOut, address[] calldata path) external {
        require(msg.sender == owner, 'Echodex: FORBIDDEN');
        feePath[tokenOut] = path;
        feePathLength[tokenOut] = path.length;
    }

    // @audit
    function calcFee(uint amountOut, address tokenOut, address pair, address factory) external view
returns (uint fee, uint feeRefund) {
        uint amountFeeTokenOut = amountOut / 1000;

        uint amountFeeRefundTokenOut = 0;
        feeRefund = 0;
        if (refundPercent[pair] > 0) {
            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund
(0.05 * 10 **18)% fee
        }

        address[] memory path = feePath[tokenOut];
        uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
        fee = amounts[amounts.length - 1];

        if (amountFeeRefundTokenOut > 0) {
            uint256[] memory amountsRefund = EchodexLibrary.getAmountsOut(factory, amountFeeRefundTo
kenOut, path);
            feeRefund = amountsRefund[amountsRefund.length - 1];
        }
    }
```

this is very problematic

when calcuting the fee, we are using spot price oracle to calculate the fee amount to the pay the protocol receiver and the refund amount,

```
uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
```

the path is set by admin

because we are using the spot price,

another malicious user can manipulate the liquidity of the pool to make the returned fee amount too large

if the attacker manipulate the liquidity to make the EchodexLibrary.getAmountsOut return value too large,

user's transaction revert

remeber:

```
(uint fee, uint feeRefund) = IEchodexFactory(factory).calcFee(amountOut, tokenOut, address(this), fa
ctory);
_payFee(fee, feeRefund, refundFeeAddress);
```

when we are calling _payFee

```
function _payFee(uint fee, uint feeRefund, address refundFeeAddress) private { //payWithTokenFee = t
rue
        address tokenFee = IEchodexFactory(factory).tokenFee();
        address receiveFeeAddress = IEchodexFactory(factory).receiveFeeAddress();
        require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');

        currentFee = currentFee - fee;

        _safeTransfer(tokenFee, receiveFeeAddress, fee - feeRefund);
        if (feeRefund > 0) {
                _safeTransfer(tokenFee, refundFeeAddress, feeRefund);
        }
}
```

if the returned fee amount too large, transaction revert in

```
require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');
```

basically a user can manipulate the liquidity of the token fee pool to make normal user's transaction revert and perform a deinal of service attack

or the attacker does not have to manipulate the liquidity, if the spot price is too high and the fee amount returned from

```
uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
```

function is too high, transaction just revert in

```
require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');
```

Recommendation

ladboy233 : Do not use spot liquidity to calculate the fee amount, instead, just transfer a percent amount of token as fee payment to receiver and a percentage of token as refund

ladboy233 : Do not use spot liquidity to calculate the fee amount, instead, just transfer a percent amount of token as fee payment to receiver and a percentage of token as refund

Client Response

Mitigated,tokenFee is our token. We will keep the price stable and set specific pairs can pay with tokenFee

ECD-12:The `amountFeeRefundTokenOut` calculation precision error in the `calcFee` function

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Xi_Zi

Code Reference

- code/contracts/EchodexFactory.sol#L72-L89
- code/contracts/EchodexFactory.sol#L77-L79

```
72:    function calcFee(uint amountOut, address tokenOut, address pair, address factory) external view returns (uint fee, uint feeRefund) {
73:        uint amountFeeTokenOut = amountOut / 1000;
74:
75:        uint amountFeeRefundTokenOut = 0;
76:        feeRefund = 0;
77:        if (refundPercent[pair] > 0) {
78:            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund (0.05 * 10 **18)% fee
79:        }
80:
81:        address[] memory path = feePath[tokenOut];
82:        uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
83:        fee = amounts[amounts.length - 1];
84:
85:        if (amountFeeRefundTokenOut > 0) {
86:            uint256[] memory amountsRefund = EchodexLibrary.getAmountsOut(factory, amountFeeRefundTokenOut, path);
87:            feeRefund = amountsRefund[amountsRefund.length - 1];
88:        }
89:    }

77:        if (refundPercent[pair] > 0) {
78:            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund (0.05 * 10 **18)% fee
79:        }
```

Description

Xi_Zi : 1. In the calcFee function of the EchodexFactory contract, when calculating amountFeeRefundTokenOut, according to the comment `// refund (0.05 * 10 **18)% fee` this should be based on the fee collected for refund, that is, it should be `amountFeeTokenOut * refundPercent[pair] / (100 * 10 ** 18);`.

2. The calculation here divides amountOut by 10 ** 18, which will cause the calculation result to be small, usually only a few wei, and there should be no precision processing here.

```
function calcFee(uint amountOut, address tokenOut, address pair, address factory) external view retu
rns (uint fee, uint feeRefund) {
        uint amountFeeTokenOut = amountOut / 1000;

        uint amountFeeRefundTokenOut = 0;
        feeRefund = 0;
        if (refundPercent[pair] > 0) {
            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund
(0.05 * 10 **18)% fee
        }

        address[] memory path = feePath[tokenOut];
        uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
        fee = amounts[amounts.length - 1];

        if (amountFeeRefundTokenOut > 0) {
            uint256[] memory amountsRefund = EchodexLibrary.getAmountsOut(factory, amountFeeRefundTo
kenOut, path);
            feeRefund = amountsRefund[amountsRefund.length - 1];
        }
    }
```

Xi_Zi : In the calcFee function of the EchodexFactory contract, if refundPercent[pair] > 0, the corresponding amountFeeRefundTokenOut will be calculated. amountOut is a value with decimals. This accuracy is the accuracy of tokenOut. When calculating here, it is problematic to use 10**18 as the calculation decimals, which may seriously cause the loss of contract funds. It needs to be calculated according to the actual accuracy of tokenOut

```
function calcFee(uint amountOut, address tokenOut, address pair, address factory) external view retu
rns (uint fee, uint feeRefund) {
        uint amountFeeTokenOut = amountOut / 1000;

        uint amountFeeRefundTokenOut = 0;
        feeRefund = 0;
        if (refundPercent[pair] > 0) {
            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund
(0.05 * 10 **18)% fee
        }

        address[] memory path = feePath[tokenOut];
        uint256[] memory amounts = EchodexLibrary.getAmountsOut(factory, amountFeeTokenOut, path);
        fee = amounts[amounts.length - 1];

        if (amountFeeRefundTokenOut > 0) {
            uint256[] memory amountsRefund = EchodexLibrary.getAmountsOut(factory, amountFeeRefundTo
kenOut, path);
            feeRefund = amountsRefund[amountsRefund.length - 1];
        }
    }
```

Recommendation

Xi_Zi : The logic needs to be reworked

Xi_Zi : It is recommended to use the actual decimals of the token for calculation:

```
amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** tokenOut.decimal());
```

Client Response

Fixed,Removed refund feature and instead of reward feature

ECD-13:Input validation in `EchodexLibrary` contract `pairFor` function

Category	Severity	Status	Contributor
Logical	Low	Acknowledged	ginlee

Code Reference

- code/contracts/libraries/EchodexLibrary.sol#L19-L37

```
    function pairFor(
        address factory,
        address tokenA,
        address tokenB
    ) internal pure returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(
            uint256(
                keccak256(
                    abi.encodePacked(
                        hex"ff",
                        factory,
                        keccak256(abi.encodePacked(token0, token1)),
                        hex"aa1ee6eba2789f443105d10c3e60662ae7e7e4389ce1ab033de342187c517846" // init code hash
                    )
                )
            )
        );
    }
```

Description

ginlee : If the factory parameter is set to address(0), which means the null address, the following will happen in the function:

The sortTokens function is called to sort tokenA and tokenB. A new pair address is created, and its calculation is based on the factory, sorted token0, token1, and a specified initialization code hash. If factory is the null address, the resulting pair address may be invalid or represent an undeployed contract address. Please note that passing the null address as the factory parameter to this function may lead to unpredictable results since it relies on these parameters to compute the pair address

Recommendation

ginlee : it is advisable to pass a valid contract address as the factory parameter to ensure the correct calculation and retrieval of the corresponding pair address. add check factory != address(0)

Client Response

Acknowledged,It will return null address when pair not exists

ECD-14:Incompatibility With Deflationary Tokens

Category	Severity	Status	Contributor
Logical	**Low**	**Mitigated**	**biakia, ladboy233**

Code Reference

- code/contracts/EchodexFarm.sol#L99-L129
- code/contracts/EchodexRouterFee.sol#L261

```
    function createPool(address tokenA, address tokenB, uint256 amountReward, address tokenReward, uint256 startDate, uint256 endDate) external {
        require(block.timestamp <= startDate, "EchodexFarm: WRONG_TIME");
        require(startDate + 30 * 60 <= endDate, "EchodexFarm: WRONG_TIME");

        address pairAddress = IEchodexFactory(factory).getPair(tokenA, tokenB);
        require(pairAddress != address(0), "EchodexFarm: PAIR_NOT_EXIST");


        uint256 amountPerSecond = amountReward.div(endDate.sub(startDate));
        pools[currentPoolId] = Pool({
            poolId: currentPoolId,
            pairAddress: pairAddress,
            amountReward: amountReward,
            tokenReward: tokenReward,
            startDate: startDate,
            endDate: endDate,
            accAmountPerShare: 0,
            totalLP: 0,
            totalReward: 0,
            amountPerSecond: amountPerSecond,
            lastRewardTimestamp: 0,
            owner: msg.sender,
            startTimeExcess: startDate,
            totalExcessReward: 0
        });

        TransferHelper.safeTransferFrom(tokenReward, msg.sender, address(this), amountReward);
        emit PoolCreated(currentPoolId, pairAddress, tokenA, tokenB, amountReward, tokenReward, startDate, endDate, amountPerSecond);

        currentPoolId++;
    }

                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
```

Description

biakia : In contract `EchodexFarm`, the function `createPool` will transfer reward tokens to the contract:

```
  TransferHelper.safeTransferFrom(tokenReward, msg.sender, address(this), amountReward);
```

If the `tokenReward` is a deflationary token, the input amount `amountReward` may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and the contract will not have enough rewards to distribute to users.

ladboy233 : In EchodexRouterFee.sol

there is this newly added logic:

```
     function _swap(
           uint256[] memory amounts,
           address[] memory path,
           address _to,
           uint[] memory amountsFeeAddMore
       ) internal virtual {
           for (uint256 i; i < path.length - 1; i++) {
               (address input, address output) = (path[i], path[i + 1]);
               (address token0,) = EchodexLibrary.sortTokens(input, output);
               uint256 amountOut = amounts[i + 1];
               (uint256 amount0Out, uint256 amount1Out) =
                   input == token0 ? (uint256(0), amountOut) : (amountOut, uint256(0));
               address to = i < path.length - 2 ? EchodexLibrary.pairFor(factory, output, path[i + 2])
  : _to;
               address pair = EchodexLibrary.pairFor(factory, input, output);
               if (amountsFeeAddMore[i] > 0) {
                   address tokenFee = IEchodexFactory(factory).tokenFee();
                   if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
                       IERC20(tokenFee).approve(pair, uint256(-1));
                   }
                   IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
                   IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
               }
               IEchodexPair(pair).swapPayWithTokenFee(amount0Out, amount1Out, to, msg.sender, new bytes
  (0));
           }
       }
```

please pay attention to the code section:

```
  if (amountsFeeAddMore[i] > 0) {
          address tokenFee = IEchodexFactory(factory).tokenFee();
          if (IERC20(tokenFee).allowance(address(this), pair) == 0) {
                  IERC20(tokenFee).approve(pair, uint256(-1));
          }
          IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
          IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
  }
```

the tokenFee (or I think the feeToken is the better name) is set by the echo dex factory admin

the contract always assume that the contract receive the exact amount of the amountsFeeAddMore[i] token

However, this is not the case if the token charge transfer fee

https://github.com/d-xo/weird-erc20#fee-on-transfer

Some tokens take a transfer fee (e.g. STA, PAXG), some do not currently charge a fee but may do so in the future (e.g. USDT, USDC).

Consider the underlying feeToken charge 1% transfer fee

the specified amount amountsFeeAddMore[i] is 100

and after IERC20(tokenFee).transferFrom is called, the contract only recever 99 amount of token

and when calling

```
  IEchodexPair(pair).addFee(amountsFeeAddMore[i]);
```

this will revert because this is calling

```
      function addFee(uint amount) external lock {

          address tokenFee = IEchodexFactory(factory).tokenFee();
          IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);

          totalFee = totalFee + amount;
          currentFee = currentFee + amount;

          emit AddFee(amount);
      }
```

the code will try to add fee with 100 amount of token, but the contract does not have 100 amount of token, the contract only receive 99 amount of token because the token charge transfer fee

Recommendation

biakia : Consider recording the actual rewards in function `createPool`:

```
uint256 beforeToken = IERC20(tokenReward).balanceOf(address(this));
TransferHelper.safeTransferFrom(tokenReward, msg.sender, address(this), amountReward);
uint256 afterToken = IERC20(tokenReward).balanceOf(address(this));
amountReward = afterToken - beforeToken;
uint256 amountPerSecond = amountReward.div(endDate.sub(startDate));
        pools[currentPoolId] = Pool({
            poolId: currentPoolId,
            pairAddress: pairAddress,
            amountReward: amountReward,
            tokenReward: tokenReward,
            startDate: startDate,
            endDate: endDate,
            accAmountPerShare: 0,
            totalLP: 0,
            totalReward: 0,
            amountPerSecond: amountPerSecond,
            lastRewardTimestamp: 0,
            owner: msg.sender,
            startTimeExcess: startDate,
            totalExcessReward: 0
        });
```

ladboy233 : Do not use fee-on-transfer token to settle the fee, or check the balanceOf before and after to determine the exact amount of token received before adding the fee,

and when adding the fee, the code should use the same balanceOf before and after check instead of assuming the contract received the amount of token in the input

Client Response

Mitigated, User need to set slippage tolerance percent higher than fee of deflationary token.

ECD-15:Unchecked ERC-20 `transferFrom()` Call

Category	Severity	Status	Contributor
Logical	**Low**	**Acknowledged**	**biakia**

Code Reference

- code/contracts/EchodexRouterFee.sol#L261
- code/contracts/EchodexRouterFee.sol#L407
- code/contracts/EchodexPair.sol#L232

```
232:        IERC20(tokenFee).transferFrom(msg.sender, address(this), amount);

261:                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);

407:                IERC20(tokenFee).transferFrom(msg.sender, address(this), amountsFeeAddMore[i]);
```

Description

biakia : In contract `EchodexRouterFee` and `EchodexPair`, the return value of the `transferFrom()` call is not checked.

Recommendation

biakia : Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Client Response

Acknowledged, tokenFee is our token. `transferFrom` in tokenFee will be ensured

ECD-16:Swap x * y = k invariant does not hold

Category	Severity	Status	Contributor
Logical	Low	Fixed	ladboy233

Code Reference

- code/contracts/EchodexPair.sol#L198
- code/contracts/EchodexPair.sol#L224

```
198:            require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reser
ve1), 'Echodex: K');

224:        require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reserve
1), 'Echodex: K');
```

Description

ladboy233 : Uniswap V2 is a established protocol, even with the presence of V3 or even V4

Uniswap V2 fork still thrive in defi

And indeed the EchoDex is a Unsiwap V2 fork as well

One of the core invariant that cannot be broken is the uniswap formula: x * y = k

How does the Uniswap make sure such invariant hold?

by using this line of code in the function swap:

https://github.com/Uniswap/v2-core/blob/ee547b17853e71ed4e0101ccfd52e70d5acded58/contracts/UniswapV2Pair.sol#L182

```
uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) :
0;
        require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**
2), 'UniswapV2: K');
        }
```

note the line code code:

```
require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'Unis
wapV2: K');
```

What is the impact if such invariant does not hold?

The whole protocol's liquidity can be drained (a $50m hack back in 2021)

this actually happens more than one times in the past on the Uniswap V2 fork:

https://medium.com/immunefi/building-a-poc-for-the-uranium-heist-ec83fbd83e9f

Let's dive a little deeper and inspect the code in snippets 1 and 2 which are from the UniswapV2 pair and Uranium pair contracts, respectively. In the original Uniswap code, we notice that a magic value of 1000 is used for mathematical operations that apply the fee to the new X and Y values after a swap, as well as in the enforcement of K. This means that all values in the K check have been scaled by an equivalent factor.

Critically, we see in Uranium's implementation that the magic value for fee calculation is 10000 instead of the original 1000. This by itself isn't an issue. The issue lies in the required statement that follows. The check does not apply the new magic value and instead uses the original 1000. This means that the K after a swap is guaranteed to be 100 times larger than the K before the swap when no token balance changes have occurred.

This also means that a bad actor can exploit this flaw and swap a minimal amount of tokens for more tokens than the algorithm would have allowed them to if implemented correctly. In this case, a bad actor can drain all liquidity in the pair contract.

Does such important invariant hold?

no

In EchodexPair, the function swap and swapPayWithTokenFee does not apply fee and consider fee when checking the k invariant

```
{ // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        require(state.balance0.mul(state.balance1) >= uint(state._reserve0).mul(state._reserve1), 'E
chodex: K');
}
```

as we can see, the original Uniswap Pair swap invariant check is:

```
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**
2), 'UniswapV2: K');
        }
```

the 1000**2 or any fee consideration is missing!

Recommendation

ladboy233 : My sincere recommendation is still mint and charge fee like the original swap code does and do not risk breaking such crucial invariant check

Client Response

Fixed, Excluded currentFee in balance before checking `K`

ECD-17:Unlocked Pragma Version

Category	Severity	Status	Contributor
Language Specific	Informational	Fixed	biakia, ginlee

Code Reference

- code/contracts/libraries/EchodexLibrary.sol#L2
- code/contracts/EchodexERC20.sol#L2

```
2:pragma solidity =0.6.6;

2:pragma solidity >=0.5.0;
```

Description

biakia : Solidity files in packages have a pragma version `>=0.5.0`. The caret `>=` points to unlocked pragma, meaning the compiler will use the specified version or above.
ginlee : The project uses multiple compiler versions with most specifying =0.6.6, some specifying >=0.5.0, <0.8.0

The dangers of allowing multiple compilers across breaking revisions are that the security bug fixes and features might differ across different contracts introducing vulnerabilities or giving a false sense of security

Recommendation

biakia : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contracts.
ginlee : Update all contracts to use pragma solidity ^0.8.0 or better a fixed version like 0.8.19, deploy with the same compiler version which was used for testing

Client Response

Fixed

ECD-18:Remove SafeMath library if use solidity version no less than 0.8

Category	Severity	Status	Contributor
Language Specific	**Informational**	**Acknowledged**	**ginlee**

Code Reference

- code/contracts/libraries/EchodexLibrary.sol#L4
- code/contracts/libraries/EchodexLibrary.sol#L9
- code/contracts/EchodexFarm.sol#L10
- code/contracts/EchodexPair.sol#L12
- code/contracts/EchodexRouterFee.sol#L12

```
4:import "./SafeMath.sol";

9:    using SafeMath for uint256;

10:    using SafeMath  for uint;

12:    using SafeMath  for uint;

12:    using SafeMath for uint256;
```

Description

ginlee : In Solidity version 0.8 and later, arithmetic operations have built-in overflow and underflow protection by default. The SafeMath library, which was commonly used in earlier versions of Solidity, is no longer necessary.

Recommendation

ginlee : remove "using SafeMath for uint" and import "./libraries/SafeMath.sol";

Client Response

Acknowledged,Smart contracts using solidity version 0.6.6

ECD-19:Consider using custom errors instead of string error message

Category	Severity	Status	Contributor
Language Specific	**Informational**	**Acknowledged**	**ginlee**

Code Reference

- code/contracts/libraries/EchodexLibrary.sol#L13
- code/contracts/libraries/EchodexLibrary.sol#L15
- code/contracts/libraries/EchodexLibrary.sol#L56

```
13:        require(tokenA != tokenB, "EchodexLibrary: IDENTICAL_ADDRESSES");

15:        require(token0 != address(0), "EchodexLibrary: ZERO_ADDRESS");

56:        require(amountA > 0, "EchodexLibrary: INSUFFICIENT_AMOUNT");
```

Description

ginlee : Solidity 0.8.4 introduced "custom errors"(https://blog.soliditylang.org/2021/04/21/custom-errors/). Custom error is better than string error message because the amount of gas depend's on the length of the string. They are more gas efficient than string error message, when it comes to deploy cost as well as runtime cost when the revert condition is met

Recommendation

ginlee : Consider using custom errors instead of revert strings Instead of

```
require(token0 != address(0), "EchodexLibrary: ZERO_ADDRESS")
```

We can use

```
error ZeroAddress();
...
...
...
revert ZeroAddress();
```

Client Response

Acknowledged, Decide not to fix it this time

ECD-20:Gas Optimization: Unnecessary value set to 0 in `EchodexPair.sol`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/contracts/EchodexPair.sol#L76-L80

```
76:    constructor() public {
77:        factory = msg.sender;
78:        totalFee = 0;
79:        currentFee = 0;
80:    }
```

Description

biakia : Since all default values in solidity are already 0, it is unnecessary to set variables to 0 in the following constructor:

```
    constructor() public {
           factory = msg.sender;
           totalFee = 0;
           currentFee = 0;
       }
```

Recommendation

biakia : Consider following fix to save gas:

```
  constructor() public {
          factory = msg.sender;
  }
```

Client Response

Fixed

ECD-21:Gas Optimization :Unused variable in `EchodexFactory.sol` and `EchodexPair.sol`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Xi_Zi, biakia

Code Reference

- code/contracts/EchodexFactory.sol#L8
- code/contracts/EchodexPair.sol#L28

```
8:    bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(EchodexPair).creationCode));

28:    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity event
```

Description

Xi_Zi : Unused variables can be removed to save gas. The INIT_CODE_PAIR_HASH variable is not used for EchodexFactory contracts, but is used to deploy scripts for some testing or debugging. It can be removed to save gas when the mainnet starts up.

```
bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(EchodexPair).creationCode));
```

biakia : In the contract `EchodexPair`, the variable `kLast` is never used. In the contract `EchodexFactory`, the variable `INIT_CODE_PAIR_HASH` is never used.

Recommendation

Xi_Zi : Remove constant variable INIT_CODE_PAIR_HASH to save gas during mainnet startup.
biakia : If these variables are not intended to be used, it is recommended to remove them to save gas.

Client Response

Fixed

ECD-22:Lack of reasonable upper boundary in `setRefundPercentPair()`

Category	Severity	Status	Contributor
Logical	Informational	Fixed	biakia

Code Reference

- code/contracts/EchodexFactory.sol#L51-L54

```
51:    function setRefundPercentPair(address pair, uint _refundPercent) external {
52:        require(msg.sender == owner, 'Echodex: FORBIDDEN');
53:        refundPercent[pair] = _refundPercent;
54:    }
```

Description

biakia : In contract `EchodexFactory`, the function is used to set refund fee for a pair:

```
function setRefundPercentPair(address pair, uint _refundPercent) external {
        require(msg.sender == owner, 'Echodex: FORBIDDEN');
        refundPercent[pair] = _refundPercent;
    }
```

The `refundPercent[pair]` will be used in function `calcFee`:

```
if (refundPercent[pair] > 0) {
            amountFeeRefundTokenOut = amountOut * refundPercent[pair]  / (100 * 10 ** 18); // refund
(0.05 * 10 **18)% fee
}
```

It is possible that the `refundPercent[pair]` is set as 100% because there is no reasonable upper boundary in function `setRefundPercentPair`.

Recommendation

biakia : Consider adding a reasonable upper boundary in function `setRefundPercentPair`.

Client Response

Fixed, Removed refund feature and instead of reward feature

ECD-23: `Initialize` function may not suitable in `EchodexPair` contract

Category	Severity	Status	Contributor
Logical	Informational	Declined	ginlee

Code Reference

- code/contracts/EchodexPair.sol#L83-L87

```
83:    function initialize(address _token0, address _token1) external {
84:        require(msg.sender == factory, 'Echodex: FORBIDDEN'); // sufficient check
85:        token0 = _token0;
86:        token1 = _token1;
87:    }
```

Description

ginlee : Initialize function often used in proxy contract for initialize contract， this contract already have a constructor, also initialize should use with initializer modifier to guarantee one time calling, but this one used with no initializer modifier

```
function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'Echodex: FORBIDDEN'); // sufficient check
        token0 = _token0;
        token1 = _token1;
}
```

Recommendation

ginlee : In this case, suggest use a common setToken function only can be called by factory(onlyFactory modifier)

Client Response

Declined, `initialize` is function needed to call when `createPair` in factory

ECD-24:Gas Optimization: No need to call `abi.encodePacked` when there's only a single bytes argument

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Xi_Zi

Code Reference

- code/contracts/EchodexFactory.sol#L8

```
8:    bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(EchodexPair).creationCode));
```

Description

Xi_Zi : In the `EchodexFactory.sol` contract, the use of `abi.encodePacked` to compute the `INIT_CODE_PAIR_HASH` unnecessarily consumes gas when there's only a single bytes argument. The expression `keccak256(abi.encodePacked(type(EchodexPair).creationCode))` can be simplified to `keccak256(type(EchodexPair).creationCode)` without impacting the functionality.

Recommendation

Xi_Zi : Replace `keccak256(abi.encodePacked(type(EchodexPair).creationCode))` with `keccak256(type(EchodexPair).creationCode)` to save some gas.

Client Response

Fixed

ECD-25:Variables that could be declared as immutable

Category	Severity	Status	Contributor
Language Specific	**Informational**	**Fixed**	**biakia**

Code Reference

- code/contracts/EchodexFarm.sol#L14
- code/contracts/EchodexPair.sol#L18

```
14:    address public factory;

18:    address public factory;
```

Description

biakia :

```
address public factory;
```

In contract `EchodexPair` and `EchodexFarm`, the linked variables assigned in the constructor can be declared as immutable. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

biakia : We recommend declaring these variables as immutable. Please note that the immutable keyword only works in Solidity version v0.6.5 and up.

```
address public immutable factory;
```

Client Response

Fixed

ECD-26:Events is missing indexed fields

Category	Severity	Status	Contributor
Language Specific	Informational	Fixed	ginlee

Code Reference

- code/contracts/EchodexFarm.sol#L43-L53
- code/contracts/EchodexFarm.sol#L55-L61
- code/contracts/EchodexFarm.sol#L63-L69

```
43:    event PoolCreated(
44:        uint256 poolId,
45:        address pairAddress,
46:        address tokenA,
47:        address tokenB,
48:        uint256 amountReward,
49:        address tokenReward,
50:        uint256 startDate,
51:        uint256 endDate,
52:        uint256 amountPerSecond
53:    );

55:    event PoolUpdate(
56:        uint256 poolId,
57:        uint256 accAmountPerShare,
58:        uint256 totalLP,
59:        uint256 totalReward,
60:        uint256 lastRewardTimestamp
61:    );

63:    event UserUpdate(
64:        address user,
65:        uint256 poolId,
66:        uint256 amount,
67:        uint256 rewardDebt,
68:        uint256 rewardEarn
69:    );
```

Description

ginlee : Index event fields make the field more quickly accessible to off-chain.Each event should use three indexed fields if there are three or more fields.

Recommendation

ginlee : Add up to three indexed fields such as

```
  event PoolCreated(
          uint256 indexed poolId,
          address indexed pairAddress,
          address tokenA,
          address tokenB,
          uint256 indexed amountReward,
          address tokenReward,
          uint256 startDate,
          uint256 endDate,
          uint256 amountPerSecond
      );
```

Client Response

Fixed

ECD-27:Gas Optimization: Remove unused imports in `EchodexLibrary`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/contracts/libraries/EchodexLibrary.sol#L5
- code/contracts/EchodexFarm.sol#L4

```
4:import './interfaces/IERC20.sol';

5:import "../interfaces/IEchodexFactory.sol";
```

Description

biakia : The contract `EchodexLibrary` includes the following unnecessary imports:

```
import "../interfaces/IEchodexFactory.sol";
```

The contract `EchodexFarm` includes the following unnecessary imports:

```
import './interfaces/IERC20.sol';
```

Recommendation

biakia : Consider removing the import statement to save on deployment gas costs.

Client Response

Fixed

ECD-28:Failure to perform zero-address checks may result in the failure of function execution.

Category	Severity	Status	Contributor
Logical	**Informational**	**Fixed**	**Xi_Zi**

Code Reference

- code/contracts/EchodexPair.sol#L106-L115

```
106:    function _payFee(uint fee, uint feeRefund, address refundFeeAddress) private { //payWithTokenFee = true
107:        address tokenFee = IEchodexFactory(factory).tokenFee();
108:        address receiveFeeAddress = IEchodexFactory(factory).receiveFeeAddress();
109:        require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');
110:
111:        currentFee = currentFee - fee;
112:        _safeTransfer(tokenFee, receiveFeeAddress, fee - feeRefund);
113:        if (feeRefund > 0) {
114:            _safeTransfer(tokenFee, refundFeeAddress, feeRefund);
115:        }
```

Description

Xi_Zi : In the EchodexPair contract _payFee function, when the tokenFee address is obtained from the factory, the tokenFee address set by the factory is not checked for zero address, and the tokenFee is not checked for zero address when the _payFee function is executed. Subsequent transfers will cause function execution failure. It is recommended to handle the case that tokenFee is zero

```
    function _payFee(uint fee, uint feeRefund, address refundFeeAddress) private { //payWithTokenFee
  = true
        address tokenFee = IEchodexFactory(factory).tokenFee();//@audit
        address receiveFeeAddress = IEchodexFactory(factory).receiveFeeAddress();
        require(currentFee >= fee, 'Echodex: INSUFFICIENT_FEE_TOKEN');

        currentFee = currentFee - fee;
        _safeTransfer(tokenFee, receiveFeeAddress, fee - feeRefund);
        if (feeRefund > 0) {
            _safeTransfer(tokenFee, refundFeeAddress, feeRefund);
        }
    }
```

Recommendation

Xi_Zi : It is recommended that the tokenFee address be treated as zero address

Client Response

Fixed

ECD-29:Gas Optimization: Redundant code in `EchodexLibrary.sol`

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Xi_Zi

Code Reference

- code/contracts/libraries/EchodexLibrary.sol#L45

```
45:        pairFor(factory, tokenA, tokenB);
```

Description

Xi_Zi : In getReserves, line #L45 is redundant and is recommended to be deleted。

```
function getReserves(
        address factory,
        address tokenA,
        address tokenB
    ) internal view returns (uint256 reserveA, uint256 reserveB) {
        (address token0, ) = sortTokens(tokenA, tokenB);
        pairFor(factory, tokenA, tokenB);//@audit
        (uint256 reserve0, uint256 reserve1, ) = IEchodexPair(pairFor(factory, tokenA, tokenB)).getR
eserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }
```

Recommendation

Xi_Zi : It is proposed to be modified as：

```
    function getReserves(address factory, address tokenA, address tokenB) internal view returns (uin
t reserveA, uint reserveB) {
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IUniswapV2Pair(pairFor(factory, tokenA, tokenB)).getReserv
es();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }
```

Client Response

Fixed

Disclaimer