Echo DEX Audit Report

Version 1

Gin

July 13, 2023

Echo DEX Audit Report

Gin

July 13, 2023

Prepared by: Gin

Table of Contents

Protocol Summary

EchoDEX is a decentralized exchange platform built on the Linea Consensys network. It is designed for fast and secure trading of cryptocurrencies, with the added benefit of being built on a trusted network.

Disclaimer

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

Risk Classification

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

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Scope

All files in codebase

Issues found

Severity	Number of Issues Found
High	0
Medium	3
Low	1
Info	0
Total	4

Findings

Medium-Risk Issues

[M-01] Potential reentrancy risk in EchodexFarm::harvest

Description:

```
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

Recommendation: 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.

[M-02] Missing contract nonce and chainId in EchodexRouterFee contract,removeLiquidityWithPermit function may lead to signature replay attacks

Description:

```
function removeLiquidityWithPermit(address tokenA, address tokenB,
   uint256 liquidity, uint256 amountAMin, uint256 amountBMin, address
   to,
uint256 deadline, bool approveMax, uint8 v, bytes32 r, bytes32 s)
   external virtual returns (uint256 amountA, uint256 amountB) {
  address pair = EchodexLibrary.pairFor(factory, tokenA, tokenB);
     uint256 value = approveMax ? uint256(-1) : liquidity;
  IEchodexPair(pair).permit(msg.sender, address(this), value, deadline,
      v, r, s);
 ....
```

```
  ....
}
```

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: 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

[M-03] Need input validation for stake function in EchodexFarm contract

Description:

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

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: 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.

Low-Risk Issues

[L-01] Input validation in EchodexLibrary contract pairFor function

Description:

```
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)),
            ),
        ),
      ),
    )
}
```

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: 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)