LoopFi PrelaunchPoints Audit Report

Version 1.0

palmcivet

May 9, 2024

LoopFi PrelaunchPoints Audit Report

palmcivet

May 8, 2024

Prepared by: palmcivet

Table of Contents

Protocol Summary

Users can lock ETH, WETH and wrapped LRTs into this contract, which will emit events tracked on a backed to calculate their corresponding amount of points. When staking, users can use a referral code encoded as `bytes32` that will give the referral extra points.

When Loop contracts are launched, the owner of the contract can call only once `setLoopAddresses` to set the `lpETH` contract as well as the staking vault for this token. This activation date is stored at `loopActivation`.

Once these addresses are set, all deposits are paused and users have `7 days` to withdraw their tokens in case they changed their mind, or they detected a malicious contract being set. On withdrawal, users loose all their points.

After these `7 days` the owner can call `convertAllETH`, that converts all ETH in the contract for `lpETH`. This conversion has the timestamp `startClaimDate`. The conversion for LRTs happens on each claim by using 0x API. This is triggered by each user.

After the global ETH conversion, users can start claiming their `lpETH` or claiming and staking them in a vault for extra rewards. The amount of `lpETH` they receive is proportional to their locked ETH amount or the amount given by the conversion by 0x API. The minimum amount to receive is determined offchain and controlled by a slippage parameter in the frontend dApp.

Disclaimer

The `palmcivet` 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

	Impact			
	Low	M	M/L	L

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

Audit Details

This was a code4rena competitive audit.

- Commit Hash: 40167e469edde09969643b6808c57e25d1b9c203

Scope

See scope.txt

Files in scope

File	Logic Contracts	Interfaces	SLOC	Purpose	Libraries used
/src/PrelaunchPoints.sol	1	****	296		@openzeppelin/contracts/token/ERC20/utils/SafeER
Totals	**1**	****	**296**		

Files out of scope

See out_of_scope.txt

File
./script/PrelaunchPoints.s.sol
./src/interfaces/ILpETH.sol

File
./src/interfaces/ILpETHVault.sol
./src/interfaces/IWETH.sol
./src/mock/AttackContract.sol
./src/mock/MockERC20.sol
./src/mock/MockLRT.sol
./src/mock/MockLpETH.sol
./src/mock/MockLpETHVault.sol
./test/PrelaunchPoints.t.sol
Totals: 10

Roles

Role	Description
Owner	Has access to privileged functions, contract owner

Executive Summary

This was my first competitive audit on Code4rena and a positive learning experience. I included the only issue I submitted and didn't look at any informational/gas optimizations in this review.

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	0
Informational	0

Severity	Number of issues found
Total	1

Findings

Medium

[M-1] Lack of slippage protection in `PrelaunchPoints::_fillQuote` can cause users to receive less funds than they are due

Description The `PrelaunchPoints` contract currently lacks explicit slippage protection in its implementation of the `PrelaunchPoints::_fillQuote` function, which handles token swaps via an external `exchangeProxy` (using the 0x protocol).

Impact This oversight can lead to potential front-running attacks where malicious actors may observe pending transactions and execute trades that capitalize on the observed trades before they are confirmed. This risk is further exacerbated because the contract relies on the integrity of externally provided `_swapCallData` without validating the minimum output amount for slippage protection. If `_swapCallData` lacks stringent conditions on the minimum output amount, the transaction is susceptible to slippage, potentially resulting in financial losses for the users of the contract.

Proof of Concepts

The `_fillQuote` function handles the swap but does not check for the actual output versus the expected minimum output.

```
1      function _fillQuote(IERC20 _sellToken, uint256 _amount, bytes
           calldata _swapCallData) internal {
2          uint256 boughtETHAmount = address(this).balance;
3          require(_sellToken.approve(exchangeProxy, _amount));
4  @>      (bool success,) = payable(exchangeProxy).call{value: 0}(
       _swapCallData);
5          if (!success) {
6              revert SwapCallFailed();
7          }
8  @>      boughtETHAmount = address(this).balance - boughtETHAmount;
9          emit SwappedTokens(address(_sellToken), _amount,
               boughtETHAmount);
10     }
```

The `_validateData` function validates input data but lacks checks for minimum received amounts in swap transactions.

Recommended Mitigation

Consider adding a `_minimumOut` parameter to the `claim`, `claimAndStake`, and `_claim` functions which is compared to an added return value of `amountReceived` from the `_fillQuote` function.

```
-    function claim(address _token, uint8 _percentage, Exchange _exchange, bytes calldata _data)
+    function claim(address _token, uint8 _percentage, Exchange _exchange, bytes calldata _data, uint256 _minimumOut)
         external
         onlyAfterDate(startClaimDate)
     {
-        _claim(_token, msg.sender, _percentage, _exchange, _data);
+        _claim(_token, msg.sender, _percentage, _exchange, _data, _minimumOut);
     }

-    function claimAndStake(address _token, uint8 _percentage, Exchange _exchange, bytes calldata _data)
+    function claimAndStake(address _token, uint8 _percentage, Exchange _exchange, bytes calldata _data, uint256 _minimumOut)
         external
         onlyAfterDate(startClaimDate)
     {
-        uint256 claimedAmount = _claim(_token, address(this), _percentage, _exchange, _data);
+        uint256 claimedAmount = _claim(_token, address(this), _percentage, _exchange, _data, _minimumOut);
         lpETH.approve(address(lpETHVault), claimedAmount);
         lpETHVault.stake(claimedAmount, msg.sender);

         emit StakedVault(msg.sender, claimedAmount);
     }

-    function _claim(address _token, address _receiver, uint8 _percentage, Exchange _exchange, bytes calldata _data)
+    function _claim(address _token, address _receiver, uint8 _percentage, Exchange _exchange, bytes calldata _data, uint256 _minimumOut)
         internal
         returns (uint256 claimedAmount)
     {
         uint256 userStake = balances[msg.sender][_token];
         if (userStake == 0) {
             revert NothingToClaim();
         }
         if (_token == ETH) {
             claimedAmount = userStake.mulDiv(totalLpETH, totalSupply);
             balances[msg.sender][_token] = 0;
             lpETH.safeTransfer(_receiver, claimedAmount);
```

```
36          } else {
37              uint256 userClaim = userStake * _percentage / 100;
38              _validateData(_token, userClaim, _exchange, _data);
39              balances[msg.sender][_token] = userStake - userClaim;
40
41              // At this point there should not be any ETH in the
                    contract
42              // Swap token to ETH
43  -           _fillQuote(IERC20(_token), userClaim, _data);
44  +           uint256 receivedAmount = _fillQuote(IERC20(_token),
    userClaim, _data);
45  +           if (receivedAmount < _minimumOut) revert
    InsufficientOutputAmount(receivedAmount, _minimumOut);
46
47              // Convert swapped ETH to lpETH (1 to 1 conversion)
48              claimedAmount = address(this).balance;
49              lpETH.deposit{value: claimedAmount}(_receiver);
50          }
51          emit Claimed(msg.sender, _token, claimedAmount);
52      }
53  .
54  .
55  .
56  -    function _fillQuote(IERC20 _sellToken, uint256 _amount, bytes
    calldata _swapCallData) internal {
57  +    function _fillQuote(IERC20 _sellToken, uint256 _amount, bytes
    calldata _swapCallData) internal returns (uint256 amountReceived) {
58          // Track our balance of the buyToken to determine how much we'
                ve bought.
59  -       uint256 boughtETHAmount = address(this).balance;
60  +       uint256 initialBalance = address(this).balance;
61
62          require(_sellToken.approve(exchangeProxy, _amount));
63
64          (bool success,) = payable(exchangeProxy).call{value: 0}(
                _swapCallData);
65          if (!success) {
66              revert SwapCallFailed();
67          }
68
69          // Use our current buyToken balance to determine how much we've
                bought.
70  -       boughtETHAmount = address(this).balance - boughtETHAmount;
71  -       emit SwappedTokens(address(_sellToken), _amount,
    boughtETHAmount);
72  +       amountReceived = address(this).balance - initialBalance;
73  +       emit SwappedTokens(address(_sellToken), _amount, amountReceived
    );
74  +       return amountReceived;
75      }
```