4th05

InterPol

Security Review Report

7 December 2024

Security Review Report

4th05

December 7, 2024

Table of Contents

Protocol Summary

InterPol is a protocol allowing users to lock their liquidity, no matter the duration, without having to renounce to the rewards possible.

InterPoL is a protocol specially designed for the Berachain ecosystem which allows token creators, launchpads, protocols, and more, to deploy protocol-owned liquidity while retaining the ability to stake it and participate in Proof of Liquidity flywheels.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

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

Overview

Contest platform	Cantina
LOC	630
Language	Solidity
Commit	d14616dad7b8d4b9b5d89f26b6843c2972e441d5
Previous audits	Pashov audit, LightChaser

Scope

- src/Beekeeper.sol
- src/HoneyLocker.sol
- src/HoneyQueen.sol

Issues found

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

Findings

[M1] Wrong amount of distributed fees when there is `_overridingReferrer`

Summary

In case of `referrerOverrides[_referrer]!=0` the amount of fees distributed to the `_overridingReferrer` is not correct because it does not take into account the `_referrerFeeShare[_referrer]` value which has been set for the OG `referrer`.

Finding Description

In the `Beekeeper::distributeFees` the `referrer` is updated considering any `_overridingReferrer`. However, getting the `FeeShare` value it passes as argument the updated `referrer` that is not linked with the `_referrerFeeShare[_referrer]` of the OG `referrer`.

```
1  function distributeFees(address _referrer, address _token, uint256
       _amount) external payable {
2          bool isBera = _token == address(0);
3          if (!isBera && _token.code.length == 0) revert NoCodeForToken()
               ;
4          // if not an authorized referrer, send everything to treasury
5          if (!isReferrer[_referrer]) {
6              isBera ? STL.safeTransferETH(treasury, _amount) : STL.
                   safeTransfer(_token, treasury, _amount);
7              emit FeesDistributed(treasury, _token, _amount);
8              return;
9          }
10         // use the referrer fee override if it exists, otherwise use
               the original referrer
11 @>       address referrer = referrerOverrides[_referrer] != address(0) ?
       referrerOverrides[_referrer] : _referrer;
12 @>       uint256 referrerFeeShareInBps = referrerFeeShare(referrer);
```

```
13          uint256 referrerFee = (_amount * referrerFeeShareInBps) /
                10000;
14
15          if (isBera) {
16              STL.safeTransferETH(referrer, referrerFee);
17              STL.safeTransferETH(treasury, _amount - referrerFee);
18          } else {
19              STL.safeTransfer(_token, referrer, referrerFee);
20              STL.safeTransfer(_token, treasury, _amount - referrerFee);
21          }
22
23          emit FeesDistributed(referrer, _token, referrerFee);
24          emit FeesDistributed(treasury, _token, _amount - referrerFee);
25      }
```

Therefore, the `Beekeeper::referrerFeeShare` function returns the `standardReferrerFeeShare` instead of the `_referrerFeeShare[_referrer]` of the OG `referrer`.

```
1   function setReferrerOverride(address _referrer, address
        _overridingReferrer) external onlyOwner {
2           if (!isReferrer[_referrer]) revert NotAReferrer();
3           referrerOverrides[_referrer] = _overridingReferrer;
4       }
5       function setReferrerFeeShare(address _referrer, uint256
            _shareOfFeeInBps) external onlyOwner {
6           if (!isReferrer[_referrer]) revert NotAReferrer();
7           _referrerFeeShare[_referrer] = _shareOfFeeInBps;
8       }
9       /*################################################################
10                              VIEW ONLY
11      ################################################################*/
12      /// @notice Returns the fee share for a given referrer
13      /// @dev If a custom fee share is set for the referrer, it returns
            that value.
14      ///      Otherwise, it returns the standard referrer fee share.
15      /// @param _referrer The address of the referrer
16      /// @return The fee share for the referrer in basis points (bps)
17      function referrerFeeShare(address _referrer) public view returns (
            uint256) {
18  @>      return _referrerFeeShare[_referrer] != 0 ? _referrerFeeShare[
            _referrer] : standardReferrerFeeShare;
19      }
```

Impact Explanation

The `_overridingReferrer` gets as `FeeShare` the `standardReferrerFeeShare` which could be greater or lower than the `_referrerFeeShare[_referrer]` would have been taken from the OG `referrer`.

Likelihood Explanation

The likelihood is high as this happens every time that `FeeShare` has to be distributed to any of the `_overridingReferrer`.

Proof of Concept

```
pragma solidity ^0.8.23;

import {Test, console} from "forge-std/Test.sol";
import {StdCheats} from "forge-std/StdCheats.sol";

import {ERC20} from "solady/tokens/ERC20.sol";
import {LibString} from "solady/utils/LibString.sol";
import {Solarray as SLA} from "solarray/Solarray.sol";

import {HoneyLocker} from "../src/HoneyLocker.sol";
import {HoneyQueen} from "../src/HoneyQueen.sol";
import {Beekeeper} from "../src/Beekeeper.sol";
import {LockerFactory} from "../src/LockerFactory.sol";
import {HoneyLockerV2} from "./mocks/HoneyLockerV2.sol";
import {GaugeAsNFT} from "./mocks/GaugeAsNFT.sol";
import {IStakingContract} from "../src/utils/IStakingContract.sol";
import {IBGT} from "./interfaces/IBGT.sol";

// prettier-ignore
contract POCTest is Test {
    using LibString for uint256;

    LockerFactory public factory;
    HoneyLocker public honeyLocker;
    HoneyLocker public newhoneyLocker;


    HoneyQueen public honeyQueen;
    Beekeeper public beekeeper;

    address public constant THJ = 0
        x4A8c9a29b23c4eAC0D235729d5e0D035258CDFA7;
    address public constant referral = address(0x5efe5a11);
    address public constant treasury = address(0x80085);
    address public constant operator = address(0xaaaa);

    string public constant PROTOCOL = "BGTSTATION";

    // These addresses are for the BARTIO network
    ERC20 public constant BGT = ERC20(0
        xbDa130737BDd9618301681329bF2e46A016ff9Ad);
    ERC20 public constant weHONEY_LP = ERC20(0
        x556b758AcCe5c4F2E1B57821E2dd797711E790F4);
    IStakingContract public weHONEY_GAUGE = IStakingContract(0
```

```
            x86DA232f6A4d146151755Ccf3e4555eadCc24cCF);
42
43      function setUp() public {
44          vm.createSelectFork("https://bartio.rpc.berachain.com/");
45
46          vm.startPrank(THJ);
47
48          beekeeper = new Beekeeper(THJ, treasury);
49          beekeeper.setReferrer(referral, true);
50
51          honeyQueen = new HoneyQueen(treasury, address(BGT), address(
                beekeeper));
52          // prettier-ignore
53          honeyQueen.setProtocolOfTarget(address(weHONEY_GAUGE), PROTOCOL
                );
54          honeyQueen.setIsSelectorAllowedForProtocol(bytes4(keccak256("
                stake(uint256)")), "stake", PROTOCOL, true);
55          honeyQueen.setIsSelectorAllowedForProtocol(bytes4(keccak256("
                withdraw(uint256)")), "unstake", PROTOCOL, true);
56          honeyQueen.setIsSelectorAllowedForProtocol(bytes4(keccak256("
                getReward(address)")), "rewards", PROTOCOL, true);
57          honeyQueen.setValidator(THJ);
58
59          factory = new LockerFactory(address(honeyQueen));
60
61          honeyLocker = HoneyLocker(payable(factory.clone(THJ, referral))
                );
62          honeyLocker.setOperator(operator);
63
64          vm.stopPrank();
65
66          vm.label(address(honeyLocker), "HoneyLocker");
67          vm.label(address(honeyQueen), "HoneyQueen");
68          vm.label(address(weHONEY_LP), "weHONEY_LP");
69          vm.label(address(weHONEY_GAUGE), "weHONEY_GAUGE");
70          vm.label(address(this), "Tests");
71          vm.label(THJ, "THJ");
72
73
74          // ---> EDIT IF NEEDED <---
75          // Deal yourself LP tokens
76          StdCheats.deal(address(weHONEY_LP), THJ, 1);
77
78      }
79
80      function test_FeesDistributedForOverrideReferrersETH() public {
81          // ETH case
82          address overridereferral = makeAddr("overridereferral");
83          vm.deal(address(beekeeper),1 ether);
84          vm.startPrank(THJ);
85          beekeeper.setStandardReferrerFeeShare(100);
```

```
86          beekeeper.setReferrerFeeShare(referral, 200);
87          beekeeper.setReferrerOverride(referral, overridereferral);
88          beekeeper.distributeFees(referral, address(0), 0.01 ether);
89          console.log(overridereferral.balance);
90          assertTrue(overridereferral.balance == 0.0002 ether, "Fee
               distributed with standardfeeshare instead of the
               referrerFeeShare");
91      }
92
93      # -- [78540] Beekeeper::distributeFees(0
           x0000000000000000000000000000005EfE5a11, 0
           x0000000000000000000000000000000000000000, 10000000000000000 [1
           e16])
94      #   # -- [0] overridereferral::fallback{value: 100000000000000}()
95      #   #   # -- [Stop]
96      #   # -- [0] 0x0000000000000000000000000000000000080085::fallback{
           value: 9900000000000000}()
97      #   #   # -- [Stop]
98      #   #-- emit FeesDistributed(recipient: overridereferral: [0
           xAc3d52AFdDe3A152E4E585560d797D16bddaFD13], token: 0
           x0000000000000000000000000000000000000000, amount:
           100000000000000 [1e14])
99      #   # -- emit FeesDistributed(recipient: 0
           x0000000000000000000000000000000000080085, token: 0
           x0000000000000000000000000000000000000000, amount:
           9900000000000000 [9.9e15])
100     #   # -- [Stop]
101     # -- [0] console::log(100000000000000 [1e14]) [staticcall]
102     #   # -- [Stop]
103     # -- [0] VM::assertTrue(false, "Fee distributed with
           standardfeeshare instead of the referrerFeeShare") [staticcall]
104     #   # -- [Revert] Fee distributed with standardfeeshare instead of
           the referrerFeeShare
105     # -- [Revert] Fee distributed with standardfeeshare instead of the
           referrerFeeShare
106
107
108 function test_FeesDistributedForOverrideReferrersERC20() public {
109         // ERC20 case
110         address overridereferral = makeAddr("overridereferral");
111         ERC20 USDCToken = ERC20(0
               xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c);
112         deal(address(USDCToken),address(beekeeper), 1 ether);
113         vm.startPrank(THJ);
114         beekeeper.setStandardReferrerFeeShare(100);
115         beekeeper.setReferrerFeeShare(referral, 200);
116         beekeeper.setReferrerOverride(referral, overridereferral);
117         beekeeper.distributeFees(referral, address(USDCToken), 0.01
               ether);
118         console.log(USDCToken.balanceOf(overridereferral));
119         assertTrue(overridereferral.balance == 0.0002 ether, "Fee
```

```
                distributed with standardfeeshare instead of the
                referrerFeeShare");
120         }
121
122         # -- [59960] Beekeeper::distributeFees(0
                x0000000000000000000000000000005EfE5a11, 0
                xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c, 10000000000000000 [1
                e16])
123         #   # -- [24782] 0xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c::
                transfer(overridereferral: [0
                xAc3d52AFdDe3A152E4E585560d797D16bddaFD13], 100000000000000 [1
                e14])
124         #   #   # -- emit Transfer(from: Beekeeper: [0
                x7f826086F7225E2C30ae5325F7aa7f56605D2d39], to: overridereferral
                : [0xAc3d52AFdDe3A152E4E585560d797D16bddaFD13], amount:
                100000000000000 [1e14])
125         #   #   # -- [Return] true
126         #   # -- [24782] 0xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c::
                transfer(0x0000000000000000000000000000000000080085,
                9900000000000000 [9.9e15])
127         #   #   # -- emit Transfer(from: Beekeeper: [0
                x7f826086F7225E2C30ae5325F7aa7f56605D2d39], to: 0
                x0000000000000000000000000000000000080085, amount:
                9900000000000000 [9.9e15])
128         #   #   # -- [Return] true
129         #   # -- emit FeesDistributed(recipient: overridereferral: [0
                xAc3d52AFdDe3A152E4E585560d797D16bddaFD13], token: 0
                xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c, amount:
                100000000000000 [1e14])
130         #   # -- emit FeesDistributed(recipient: 0
                x0000000000000000000000000000000000080085, token: 0
                xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c, amount:
                9900000000000000 [9.9e15])
131         #   # -- [Stop]
132         # -- [558] 0xd6D83aF58a19Cd14eF3CF6fe848C9A4d21e5727c::balanceOf(
                overridereferral: [0xAc3d52AFdDe3A152E4E585560d797D16bddaFD13])
                [staticcall]
133         #   # -- [Return] 100000000000000 [1e14]
134         # -- [0] console::log(100000000000000 [1e14]) [staticcall]
135         #   # -- [Stop]
136         # -- [0] VM::assertTrue(false, "Fee distributed with
                standardfeeshare instead of the referrerFeeShare") [staticcall]
137         #   # -- [Revert] Fee distributed with standardfeeshare instead of
                the referrerFeeShare
138         # -- [Revert] Fee distributed with standardfeeshare instead of the
                referrerFeeShare
139     }
```

Recommendation

```
function distributeFees(address _referrer, address _token, uint256 _amount) external payable {
        bool isBera = _token == address(0);
        if (!isBera && _token.code.length == 0) revert NoCodeForToken();
        // if not an authorized referrer, send everything to treasury
        if (!isReferrer[_referrer]) {
            isBera ? STL.safeTransferETH(treasury, _amount) : STL.safeTransfer(_token, treasury, _amount);
            emit FeesDistributed(treasury, _token, _amount);
            return;
        }
        // use the referrer fee override if it exists, otherwise use the original referrer
        address referrer = referrerOverrides[_referrer] != address(0) ? referrerOverrides[_referrer] : _referrer;
-       uint256 referrerFeeShareInBps = referrerFeeShare(referrer);
+       uint256 referrerFeeShareInBps = referrerFeeShare(_referrer);

        uint256 referrerFee = (_amount * referrerFeeShareInBps) / 10000;

        if (isBera) {
            STL.safeTransferETH(referrer, referrerFee);
            STL.safeTransferETH(treasury, _amount - referrerFee);
        } else {
            STL.safeTransfer(_token, referrer, referrerFee);
            STL.safeTransfer(_token, treasury, _amount - referrerFee);
        }

        emit FeesDistributed(referrer, _token, referrerFee);
        emit FeesDistributed(treasury, _token, _amount - referrerFee);
    }
```