RebateFi Hook - Initial Findings Report

Version 0.1

HENRY LUBIGILI

December 9, 2025

RebateFi Hook - Findings Report

HENRY LUBIGILI

December 9, 2025

RebateFi Hook - Findings Report

Prepared by: HENRY LUBIGILI

Lead Auditor:

- HENRY LUBIGILI

Assisting Auditors:

- None

Table of contents

See table

Disclaimer

I Henry made all effort to find the vulnerabilities in the code in the given time period, but holds no responsibilities for the the findings provided in this document. A security audit 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

Audit Details

Scope

```
./src/
  -- ReFi.sol
  -- RebateFiHook.sol
```

Protocol Summary

RebateFi Hook is a Uniswap V4 hook implementation that enables asymmetric fee structures for designated ReFi (Rebate Finance) tokens. The protocol applies differential LP fee overrides based on swap direction, creating an innovative economic model that incentivizes token accumulation while managing sell pressure.

Roles

There are 2 main actors in this protocol:

1. owner: RESPONSIBILITIES:

- deploys hook contract with designated ReFi token address
- can modify buy and sell fee rates via ChangeFee() function
- can withdraw accumulated tokens from hook contract to specified addresses
- has full administrative control over fee configuration
- monitors protocol revenue and token accumulation

2. swapper: RESPONSIBILITIES:

- can execute swaps through Uniswap V4 pools utilizing this hook
- can buy ReFi tokens with reduced or zero fees (depending on configuration)
- can sell ReFi tokens subject to configured sell fees
- must provide sufficient token approvals for swaps

LIMITATIONS:

- cannot bypass hook-enforced fees
- cannot modify fee structures
- subject to pool liquidity constraints
- must use pools that contain the designated ReFi token

Executive Summary

Issues found

Severity	Number of issues found
High	1
Medium	0
Low	1
Info	0
Total	2

Findings

High

[H-1] Incorrect Pool Validation in `ReFiSwapRebateHook::_beforeInitialize` Causes `ReFi` Pools to Fail When Token Is `currency0`

Description: The `ReFiSwapRebateHook::_beforeInitialize` function is meant to ensure that the `ReFi` token is part of a pool before initialization. However, the condition mistakenly checks `currency1` twice and ignores `currency0`. This logic error means that any pool where `ReFi` is positioned as `currency0` will always revert, even though the pool is valid. Because pool initialization is a core function, this bug directly blocks half of all possible pool configurations, making the hook unusable in many legitimate scenarios. The issue is deterministic, easily triggered, and results in wasted gas, failed deployments, and restricted functionality. Its severity is high because it breaks expected behavior in a fundamental workflow and prevents the system from supporting valid pools.

```
1  function _beforeInitialize(address, PoolKey calldata key, uint160)
       internal view override returns (bytes4) {
2   @>       if (Currency.unwrap(key.currency1) != ReFi && Currency.unwrap
       (key.currency1) != ReFi) {
3          if (Currency.unwrap(key.currency1) != ReFi && Currency.unwrap(
               key.currency1) != ReFi) {
4              revert ReFiNotInPool();
5          }
6
7          return BaseHook.beforeInitialize.selector;
8      }
```

Impact: Users attempting to initialize pools with `ReFi` positioned as `currency0` will always face transaction failures. This results in wasted gas costs, failed deployments, and frustration, as valid pools are consistently rejected. The deterministic nature of the bug means users cannot bypass or

work around it, directly reducing confidence in the system and discouraging participation. For the contract, the flawed validation logic prevents half of all legitimate pool configurations from being supported. This restricts flexibility, undermines interoperability with other protocols, and breaks a fundamental workflow in pool initialization. Because the error is guaranteed to occur whenever `ReFi` is in `currency0`, the issue imposes both functional and economic costs, making it a high-severity vulnerability that impacts core usability and trust in the protocol.

Proof of Concept:

The code demonstrates that `RebateFiHook::_beforeInitialize` incorrectly rejects pools whenever ReFi is positioned as currency1, causing valid pools to fail initialization and imposing high-severity functional and economic costs.

Proof of Code:

Code

Place the below code into the `RebateFiHookTest.t.sol` file.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import {Test} from "forge-std/Test.sol";
import {ReFiSwapRebateHook} from "../src/RebateFiHook.sol";
import {Currency, CurrencyLibrary} from "v4-core/types/Currency.sol";
import "./mocks/MockPoolManager.sol"; // path to your mock
import {PoolKey} from "v4-core/types/PoolKey.sol";
import {IHooks} from "v4-core/interfaces/IHooks.sol";

contract TestRefiHook is Test {
    ReFiSwapRebateHook public hook;
    address public owner = address(1);
    address public reFiToken = address(0
        x1234567890123456789012345678901234);
    address public token = address(0x9876543210987654321098765432109876
        );
    IPoolManager public poolManager = IPoolManager(address(0
        x1111111111111111111111111111111111));

    function setUp() public {
        vm.prank(owner);
        hook = new ReFiSwapRebateHook(poolManager, reFiToken);
    }

    function testCurrencyValidation() public {
        PoolKey memory validKey = PoolKey({
            currency0: Currency.wrap(reFiToken),
            currency1: Currency.wrap(token),
            hooks: IHooks(address(hook)),
            fee: 3000,
```

```
            tickSpacing: 60
        });

        vm.prank(owner);
        bytes4 selector = hook.beforeInitialize(address(0), validKey,
            0);
        assertEq(selector, hook.beforeInitialize.selector);

        PoolKey memory invalidKey = PoolKey({
            currency0: Currency.wrap(token),
            currency1: Currency.wrap(reFiToken),
            hooks: IHooks(address(hook)),
            fee: 3000,
            tickSpacing: 60
        });

        vm.expectRevert(bytes("ReFiNotInPool"));
        vm.prank(owner);
        hook.beforeInitialize(address(0), invalidKey, 0);
    }
}
```

Recommended Mitigation

Correct the validation logic: Update the `ReFiSwapRebateHook::_beforeInitialize` function to check both `currency0` and `currency1` instead of checking `currency1` twice.

```
    function _beforeInitialize(address, PoolKey calldata key, uint160)
        internal view override returns (bytes4) {
-       if (Currency.unwrap(key.currency1) != ReFi && Currency.unwrap(
    key.currency1) != ReFi) {
+       if (Currency.unwrap(key.currency0) != ReFi && Currency.unwrap(
    key.currency1) != ReFi) {
            revert ReFiNotInPool();
        }

        return BaseHook.beforeInitialize.selector;
    }
```

Low

[L-1]. `ReFiSwapRebateHook::TokensWithdrawn` event has parameters out of order

Impact:

Event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Description

When the `TokensWithdrawn` event is emitted in the `ReFiSwapRebateHook::withdrawTokens` function, it logs values in an incorrect order. The `token` should go in the first parameter position, whereas the `to` should go in the second parameter position.

```
1      function withdrawTokens(address token, address to, uint256 amount)
           external onlyOwner {
2          IERC20(token).transfer(to, amount);
3  @>      emit TokensWithdrawn(to, token , amount);
4      }
```

Recommended Mitigation

emit the event with parameters in an appropriate positions.

```
1  - emit TokensWithdrawn(to, token , amount);
2  + event TokensWithdrawn(address indexed token, address indexed to,
       uint256 amount);
```