Fee Override Hooklet Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Giovanni Di Siena

Pontifex

July 19, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse 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.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

The `FeeOverrideHooklet` is a contract adjacent to the core Bunni v2 protocol that allows for dynamic fee adjustments to be made by the pool owner. This feature is particularly useful for managing the pool's fee structure in response to market conditions or other external factors.

5 Audit Scope

The scope of this audit is limited to `src/FeeOverrideHooklet.sol`, although the deployment script was also reviewed to ensure that the hooklet permissions are encoded correctly.

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the Fee Override Hooklet smart contracts provided by Bacon Labs. In this period, a total of 6 issues were found.

The following points were considered when evaluating the correctness of this hooklet:

- Does `msg.sender` need to use multicaller context?
- Is BunniToken ownership/revocation correct?
- Does the hooklet need `BunniHub`/`BunniHook` access controls?
- Are any other interface implementations needed?
- Can there be issues when the overridden fee is either bound?
- Can there be issues when the overridden price is `false`/`0`?
- Are the deployment flags correct?

- Are there any relevant findings from previous audits?

While this surfaced only informational findings related to the hooklet's functionality, it did aid in identifying issues in the core Bunni contracts related to insufficient clamping of overridden hooklet fee/prices and a mismatch between the quoted and actual swap fee amounts.

The test suite is very minimal; however, use of this hooklet within the more comprehensive core Bunni test suite failed to produce any failures.

Summary

Project Name	Fee Override Hooklet
Repository	hooklets
Commit	51e1c1d896ac...
Audit Timeline	Jul 11th - Jul 14th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	4
Gas Optimizations	0
Total Issues	6

Summary of Findings

[M-1] Inconsistent effective swap fee calculations between `BunniQuoter::quoteSwap` and `BunniHookLogic::beforeSwap` due to incorrect application of hook and curator fees	Resolved
[L-1] Insufficient clamping in `HookletLib::hookletBeforeSwap` can result in unexpected reverts	Resolved
[I-1] Lack of multicall support for `FeeOverrideHooklet::setFeeOverride`	Resolved
[I-2] Outdated reference to rebalance in `IHooklet::afterSwap` should be removed	Resolved
[I-3] `BeforeSwapFeeOverride` struct could be used in the `FeeOverride` struct	Acknowledged
[I-4] No need to bound the fee override in `FeeOverrideHooklet::setFeeOverride` as it is clamped in `HookletLib::beforeSwap`	Acknowledged

7 Findings

7.1 Medium Risk

7.1.1 Inconsistent effective swap fee calculations between `BunniQuoter::quoteSwap` and `BunniHookLogic::beforeSwap` due to incorrect application of hook and curator fees

Description: While both `BunniQuoter::quoteSwap` and `BunniHookLogic::beforeSwap` are intended to contain the same fee logic, there remain a couple of inconsistencies that appear to have either been missed or reintroduced as a regression.

Firstly, `BunniQuoter::quoteSwap` applies an additional adjustment to the swap fee when `useAmAmmFee` is enabled to add the hook fee, increasing the effective fee charged to the swapper:

```
swapFee += uint24(hookFeesBaseSwapFee.mulDivUp(hookFeeModifier, MODIFIER_BASE));
```

This adjustment is intended to account for the actual cost of the swap including the hook fee; however, in `BunniHookLogic::beforeSwap`, which performs the actual execution of the swap, this adjustment is not actually applied. The base fee is computed for use in subsequent event emission and passed to the `HookletLib::hookletAfterSwap` call without including the hook fee, resulting in an incorrect effective swap fee being used during execution.

Secondly, while `BunniQuoter::quoteSwap` intends to provide pricing and fee estimates for a simulated swap, it omits the application of a curator fee that is present and deducted alongside the protocol and am-AMM fees in the actual `BunniHookLogic::beforeSwap` logic:

```
curatorFeeAmount = baseSwapFeeAmount.mulDivUp(curatorFees.feeRate, CURATOR_FEE_BASE);
```

As such, this represents an additional inconsistency between these two implementations that results in the quote differing from the actual execution, in this case charging more than quoted. This portion of the discrepancy originates from the design of `BunniQuoter::quoteSwap` as a view function that does not have direct access to the `curatorFees` in `HookStorage`.

Impact: Off-chain components relying on the `Swap` event may reference an incorrect value for the swap fee. Interfaces relying on `BunniQuoter::quoteSwap` to estimate the swap fees/output will underestimate the total fee and overestimate the user's output. Hooklets relying on `swapFee` for accounting or additional fee calculations may charge an incorrect amount. When the hook fee modifier and curator fee rate are non-zero, the swap fee used in the actual deduction and emitted in events is smaller than that computed by the quote.

Proof of Concept: The following fuzz test can be added to `BunniHook.t.sol` to assert equivalence, and demonstrates that this is not currently the case:

```
function test_fuzz_quoter_quoteSwap(
    uint256 swapAmount,
    bool zeroForOne,
    bool amAmmEnabled
) external {
    swapAmount = bound(swapAmount, 1e6, 1e36);

    // deploy mock hooklet with all flags
    bytes32 salt;
    unchecked {
        bytes memory creationCode = type(HookletMock).creationCode;
        for (uint256 offset; offset < 100000; offset++) {
            salt = bytes32(offset);
            address deployed = computeAddress(address(this), salt, creationCode);
            if (
                uint160(bytes20(deployed)) & HookletLib.ALL_FLAGS_MASK == HookletLib.ALL_FLAGS_MASK
                    && deployed.code.length == 0
            ) {
                break;
            }
        }
    }
    HookletMock hooklet = new HookletMock{salt: salt}();

    ILiquidityDensityFunction ldf_ =
        new UniformDistribution(address(hub), address(bunniHook), address(quoter));
    bytes32 ldfParams = bytes32(abi.encodePacked(ShiftMode.STATIC, int24(-5) * TICK_SPACING, int24(5) *
        ↪  TICK_SPACING));

    (IBunniToken bunniToken, PoolKey memory key) = hub.deployBunniToken(
        IBunniHub.DeployBunniTokenParams({
            currency0: Currency.wrap(address(token0)),
            currency1: Currency.wrap(address(token1)),
            tickSpacing: TICK_SPACING,
            twapSecondsAgo: 0,
            liquidityDensityFunction: ldf_,
            hooklet: hooklet,
            ldfType: LDFType.STATIC,
            ldfParams: ldfParams,
            hooks: bunniHook,
            hookParams: abi.encodePacked(
                FEE_MIN,
                FEE_MAX,
                FEE_QUADRATIC_MULTIPLIER,
                FEE_TWAP_SECONDS_AGO,
                POOL_MAX_AMAMM_FEE,
                SURGE_HALFLIFE,
                SURGE_AUTOSTART_TIME,
                VAULT_SURGE_THRESHOLD_0,
                VAULT_SURGE_THRESHOLD_1,
                REBALANCE_THRESHOLD,
                REBALANCE_MAX_SLIPPAGE,
                REBALANCE_TWAP_SECONDS_AGO,
                REBALANCE_ORDER_TTL,
                amAmmEnabled,
                ORACLE_MIN_INTERVAL,
                uint48(1)
            ),
            vault0: ERC4626(address(0)),
            vault1: ERC4626(address(0)),
            minRawTokenRatio0: 0.08e6,
            targetRawTokenRatio0: 0.1e6,
            maxRawTokenRatio0: 0.12e6,
            minRawTokenRatio1: 0.08e6,
            targetRawTokenRatio1: 0.1e6,
            maxRawTokenRatio1: 0.12e6,
            sqrtPriceX96: TickMath.getSqrtPriceAtTick(4),
            name: bytes32("BunniToken"),
            symbol: bytes32("BUNNI-LP"),
            owner: address(this),
            metadataURI: "metadataURI",
            salt: ""
        })
    );

    // make initial deposit to avoid accounting for MIN_INITIAL_SHARES
    uint256 depositAmount0 = PRECISION;
    uint256 depositAmount1 = PRECISION;
    vm.startPrank(address(0x6969));
    token0.approve(address(PERMIT2), type(uint256).max);
    token1.approve(address(PERMIT2), type(uint256).max);
    weth.approve(address(PERMIT2), type(uint256).max);
    PERMIT2.approve(address(token0), address(hub), type(uint160).max, type(uint48).max);
```

```
PERMIT2.approve(address(token1), address(hub), type(uint160).max, type(uint48).max);
PERMIT2.approve(address(weth), address(hub), type(uint160).max, type(uint48).max);
vm.stopPrank();

_makeDepositWithFee(key, depositAmount0, depositAmount1, address(0x6969), 0, 0, "");

vm.prank(HOOK_FEE_RECIPIENT_CONTROLLER);
bunniHook.setHookFeeRecipient(HOOK_FEE_RECIPIENT);
bunniHook.setHookFeeModifier(HOOK_FEE_MODIFIER);
bunniHook.curatorSetFeeRate(key.toId(), uint16(MAX_CURATOR_FEE));

if (amAmmEnabled) {
    _makeDeposit(key, 1_000_000_000 * depositAmount0, 1_000_000_000 * depositAmount1,
    ↪  address(this), "");

    bunniToken.approve(address(bunniHook), type(uint256).max);
    bunniHook.bid({
        id: key.toId(),
        manager: address(this),
        payload: bytes6(bytes3(POOL_MAX_AMAMM_FEE)),
        rent: 1e18,
        deposit: uint128(K) * 1e18
    });

    skipBlocks(K);

    IAmAmm.Bid memory bid = bunniHook.getBid(key.toId(), true);
    assertEq(bid.manager, address(this), "manager incorrect");
}

(Currency inputToken, Currency outputToken) =
    zeroForOne ? (key.currency0, key.currency1) : (key.currency1, key.currency0);
_mint(inputToken, address(this), swapAmount * 2);
IPoolManager.SwapParams memory params = IPoolManager.SwapParams({
    zeroForOne: zeroForOne,
    amountSpecified: -int256(swapAmount),
    sqrtPriceLimitX96: zeroForOne ? TickMath.MIN_SQRT_PRICE + 1 : TickMath.MAX_SQRT_PRICE - 1
});

// quote swap
(bool success,,, uint256 inputAmount, uint256 outputAmount, uint24 swapFee,) =
↪  quoter.quoteSwap(address(this), key, params);
assertTrue(success, "quoteSwap failed");

// execute swap
uint256 actualInputAmount;
uint256 actualOutputAmount;
{
    uint256 beforeInputTokenBalance = inputToken.balanceOfSelf();
    uint256 beforeOutputTokenBalance = outputToken.balanceOfSelf();
    vm.recordLogs();
    swapper.swap(key, params, type(uint256).max, 0);
    actualInputAmount = beforeInputTokenBalance - inputToken.balanceOfSelf();
    actualOutputAmount = outputToken.balanceOfSelf() - beforeOutputTokenBalance;
}

// check if swapFee matches
Vm.Log[] memory logs = vm.getRecordedLogs();
Vm.Log memory swapLog;
for (uint256 i = 0; i < logs.length; i++) {
    if (logs[i].emitter == address(bunniHook) && logs[i].topics[0] ==
    ↪  keccak256("Swap(bytes32,address,bool,bool,uint256,uint256,uint160,int24,uint24,uint256)")) {
        swapLog = logs[i];
```

```
            break;
        }
    }

    // parse log and extract swapFee from calldata
    (,,,,,,uint24 fee,) = abi.decode(swapLog.data, (
        bool,
        bool,
        uint256,
        uint256,
        uint160,
        int24,
        uint24,
        uint256
    ));

    assertEq(swapFee, fee, "swapFee doesn't match quoted swapFee");

    // check if actual amounts match quoted amounts
    assertEq(actualInputAmount, inputAmount, "actual input amount doesn't match quoted input amount");
    assertEq(actualOutputAmount, outputAmount, "actual output amount doesn't match quoted output
    ↪   amount");
}
```

Recommended Mitigation: Ensure that the hook fee modifier and curator fee logic is consistently applied across both `BunniQuoter::quoteSwap` and `BunniHookLogic::beforeSwap`. To do so, it may be necessary to expose a view function to query the curator fee rate per pool.

The `outputAmount`/`swapFeeAmount` adjustments are confusing because the logic is slightly inconsistent which makes them difficult to compare. If possible, consider pulling out this common shared logic into a separate library function to minimize the likelihood of introducing further inconsistencies.

Bacon Labs: Fixed in commits 2a54265 and 99dd8d4.

Cyfrin: Verified. The quoter and hook implementations are now consistent.

7.2 Low Risk

7.2.1 Insufficient clamping in `HookletLib::hookletBeforeSwap` can result in unexpected reverts

Description: `HookletLib::hookletBeforeSwap` and `HookletLib::hookletBeforeSwapView` decode the fee and price override data returned from external hooklet calls, applying a clamping operation on these values to ensure that they lie within specified bounds:

```
// clamp the override values to the valid range
fee = feeOverridden ? uint24(fee.clamp(0, SWAP_FEE_BASE)) : 0;
sqrtPriceX96 =
    priceOverridden ? uint160(sqrtPriceX96.clamp(TickMath.MIN_SQRT_PRICE, TickMath.MAX_SQRT_PRICE)) : 0;
```

However, the following edge cases have not been accounted for:

- Use of `SWAP_FEE_BASE` as the inclusive upper bound contradicts validation elsewhere throughout the codebase, for example in `BunniHookLogic::isValidParams` and `FeeOverrideHooklet::setFeeOverride` where a fee equal to `SWAP_FEE_BASE` is explicitly prevented. Allowing this value may cause a division-by-zero error in downstream logic such as `BunniHookLogic::beforeSwap` which relies on `SWAP_FEE_BASE - fee` to be non-zero.
- Bounding the overridden `sqrtPriceX96` to the range `[TickMath.MIN_SQRT_PRICE, TickMath.MAX_SQRT_PRICE]` fails to consider the sqrt price corresponding to the range of usable ricks $r_{\text{min}} = \lfloor \frac{t_{\text{min}}}{w} \rfloor \cdot w$ and $r_{\text{max}} = \left(\lfloor \frac{t_{\text{max}}}{w} \rfloor - 1\right) \cdot w$ as defined in the whitepaper. Allowing prices outside of this range may cause reverts due to the use of invalid ticks when `getSqrtPriceAtTick()` is called.

Impact: If the fee override value equals `SWAP_FEE_BASE`, this may result in a revert due to division-by-zero during further swap execution in `BunniHookLogic::beforeSwap`. If the sqrt price is such that the corresponding tick lies outside the range of usable ricks then execution will similarly revert within this invocation.

Proof of Concept: Apply the following patch:

```
---
 test/BunniHook.t.sol | 110 +++++++++++++++++-------------------------
 1 file changed, 41 insertions(+), 65 deletions(-)

diff --git a/test/BunniHook.t.sol b/test/BunniHook.t.sol
index 0a7978bd..25f3eb08 100644
--- a/test/BunniHook.t.sol
+++ b/test/BunniHook.t.sol
@@ -8,6 +8,9 @@ import {IAmAmm} from "biddog/interfaces/IAmAmm.sol";
 import "./BaseTest.sol";
 import {BunniStateLibrary} from "../src/lib/BunniStateLibrary.sol";

+import {SWAP_FEE_BASE} from "src/base/Constants.sol";
+import {CustomRevert} from "@uniswap/v4-core/src/libraries/CustomRevert.sol";
+
 contract BunniHookTest is BaseTest {
     using TickMath for *;
     using FullMathX96 for *;
@@ -1388,12 +1391,6 @@ contract BunniHookTest is BaseTest {
         ldf_.setMinTick(-30);

         // deploy pool with hooklet
-        // this should trigger:
-        // - before/afterInitialize
-        // - before/afterDeposit
-        uint24 feeMin = 0.3e6;
-        uint24 feeMax = 0.5e6;
-        uint24 feeQuadraticMultiplier = 1e6;
         (Currency currency0, Currency currency1) = (Currency.wrap(address(token0)),
         ↪  Currency.wrap(address(token1)));
         (IBunniToken bunniToken, PoolKey memory key) = _deployPoolAndInitLiquidity(
             currency0,
@@ -1401,11 +1398,12 @@ contract BunniHookTest is BaseTest {
             ERC4626(address(0)),
             ERC4626(address(0)),
             ldf_,
+            IHooklet(hooklet),
             ldfParams,
             abi.encodePacked(
-                feeMin,
-                feeMax,
-                feeQuadraticMultiplier,
+                uint24(0.3e6),
+                uint24(0.5e6),
+                uint24(1e6),
                 FEE_TWAP_SECONDS_AGO,
                 POOL_MAX_AMAMM_FEE,
                 SURGE_HALFLIFE,
@@ -1419,68 +1417,46 @@ contract BunniHookTest is BaseTest {
                 true, // amAmmEnabled
                 ORACLE_MIN_INTERVAL,
                 MIN_RENT_MULTIPLIER
-            )
+            ),
+            bytes32("")
         );
-        address depositor = address(0x6969);
-
-        // transfer bunniToken
-        // this should trigger:
-        // - before/afterTransfer
-        address recipient = address(0x8008);
-        vm.startPrank(depositor);
-        bunniToken.transfer(recipient, bunniToken.balanceOf(depositor));
-        vm.stopPrank();
-        vm.startPrank(recipient);
-        bunniToken.transfer(depositor, bunniToken.balanceOf(recipient));
-        vm.stopPrank();

-        // withdraw liquidity
-        // this should trigger:
-        // - before/afterWithdraw
-        vm.startPrank(depositor);
-        hub.withdraw(
-            IBunniHub.WithdrawParams({
-                poolKey: key,
-                recipient: depositor,
-                shares: bunniToken.balanceOf(depositor),
-                amount0Min: 0,
-                amount1Min: 0,
-                deadline: block.timestamp,
-                useQueuedWithdrawal: false
-            })
-        );
-        vm.stopPrank();
-
-        // shift LDF to trigger rebalance during the next swap
-        ldf_.setMinTick(-20);
-
-        // make swap
-        // this should trigger:
-        // - before/afterSwap
+        // make swap to trigger beforeSwap
        _mint(currency0, address(this), 1e6);
        IPoolManager.SwapParams memory params = IPoolManager.SwapParams({
-            zeroForOne: true,
-            amountSpecified: -int256(1e6),
-            sqrtPriceLimitX96: TickMath.MIN_SQRT_PRICE + 1
+            zeroForOne: false,
+            amountSpecified: int256(1e6),
+            sqrtPriceLimitX96: TickMath.MAX_SQRT_PRICE - 1
        });
-        vm.recordLogs();
-        _swap(key, params, 0, "");

-        // fill rebalance order
-        // this should trigger:
-        // - afterRebalance
-        Vm.Log[] memory logs = vm.getRecordedLogs();
-        Vm.Log memory orderEtchedLog;
-        for (uint256 i = 0; i < logs.length; i++) {
-            if (logs[i].emitter == address(floodPlain) && logs[i].topics[0] ==
↪  IOnChainOrders.OrderEtched.selector) {
-                orderEtchedLog = logs[i];
-                break;
-            }
-        }
-        IFloodPlain.SignedOrder memory signedOrder = abi.decode(orderEtchedLog.data,
↪  (IFloodPlain.SignedOrder));
-        IFloodPlain.Order memory order = signedOrder.order;
-        _mint(key.currency0, address(this), order.consideration.amount);
-        floodPlain.fulfillOrder(signedOrder);
+        hooklet.setBeforeSwapOverride(true, uint24(SWAP_FEE_BASE), false, uint24(0));
+        vm.expectRevert(
+            abi.encodeWithSelector(
+                CustomRevert.WrappedError.selector,
+                key.hooks,
+                BunniHook.beforeSwap.selector,
+                abi.encodePacked(bytes4(keccak256("MulDivFailed()"))),
+                abi.encodePacked(bytes4(keccak256("HookCallFailed()")))
+            )
+        );
+        _swap(key, params, 0, "");
+
+        int24 tickAtPrice = TickMath.MIN_TICK;
+        uint160 priceOverride = TickMath.getSqrtPriceAtTick(tickAtPrice);
+        hooklet.setBeforeSwapOverride(false, uint24(0), true, priceOverride);
+        vm.expectRevert(
+            abi.encodeWithSelector(
+                CustomRevert.WrappedError.selector,
+                key.hooks,
+                BunniHook.beforeSwap.selector,
+                abi.encodeWithSelector(
+                    TickMath.InvalidTick.selector,
+                    tickAtPrice - ((tickAtPrice % key.tickSpacing + key.tickSpacing) % key.tickSpacing)
+                ),
+                abi.encodePacked(bytes4(keccak256("HookCallFailed()")))
+            )
+        );
+        _swap(key, params, 0, "");
    }
--
2.40.0
```

Recommended Mitigation: Update the clamping logic in both `HookletLib::hookletBeforeSwap` and `Hook-`

letLib::hookletBeforeSwapView to subtract one from SWAP_FEE_BASE to prevent allowing this as a valid upper bound and align the behavior with other instances where similar validation is already performed:

```
- fee = feeOverridden ? uint24(fee.clamp(0, SWAP_FEE_BASE)) : 0;
+ fee = feeOverridden ? uint24(fee.clamp(0, SWAP_FEE_BASE - 1)) : 0;
```

Additionally prevent the overridden price from being specified as corresponding to a tick that is outside the range of usable ricks:

```
- sqrtPriceX96 =
-   priceOverridden ? uint160(sqrtPriceX96.clamp(TickMath.MIN_SQRT_PRICE, TickMath.MAX_SQRT_PRICE)) : 0;
+ sqrtPriceX96 =
+   priceOverridden ? uint160(sqrtPriceX96.clamp(TickMath.getSqrtPriceAtTick((TickMath.MIN_TICK /
↪   key.tickSpacing) * key.tickSpacing), TickMath.getSqrtPriceAtTick(((TickMath.MAX_TICK /
↪   key.tickSpacing) - 1) * key.tickSpacing))) : 0;
```

Bacon Labs: Fixed in commits 45643ef and 0b5a708.

Cyfrin: Verified. The fee clamping logic has been modified to prevent SWAP_FEE_BASE from being used. The price clamping logic has also been modified to bound the override between the sqrt prices corresponding to the minimum and maximum usable ticks.

7.3 Informational

7.3.1 Lack of multicall support for `FeeOverrideHooklet::setFeeOverride`

Description: `FeeOverrideHooklet::setFeeOverride` relies on `msg.sender` when validating ownership of the corresponding `BunniToken`; however, this will be incorrect when invoked through a multicaller contract and cause validation to fail even if the original caller is the actual owner.

Given that any account is free to deploy a Bunni pool and `LibMulticaller` is used heavily throughout the core Bunni contracts, it may be desirable to allow the owner of the pool to perform batched actions on both the hooklet and Bunni itself.

Impact: Any legitimate pool owner interacting with the `FeeOverrideHooklet` via a multicaller contract will be incorrectly prevented from doing so, potentially breaking external compatibilities that rely on multicall support.

Recommended Mitigation: Replace direct usage of `msg.sender` with `LibMulticaller::senderOrSigner` to remain consistent with the core Bunni contracts. This approach preserves compatibility with both direct and batched calls, ensuring proper access control while supporting multicall infrastructure.

Bacon Labs: Fixed in commit 9cf16a8.

Cyfrin: Verified. `FeeOverrideHooklet::setFeeOverride` is now compatible with multicall invocations.

7.3.2 Outdated reference to rebalance in `IHooklet::afterSwap` should be removed

Description: There is an outdated reference to rebalance in the `IHooklet::afterSwap` dev comment which should be removed given `IHooklet::afterRebalance` has since been added:

```
    /// @notice Called after a swap operation.
@>  /// @dev Also called after a rebalance order execution, in which case returnData will only have
@>  /// inputAmount and outputAmount filled out.
    /// @param sender The address of the account that initiated the swap.
    /// @param key The Uniswap v4 pool's key.
    /// @param params The swap's input parameters.
    /// @param returnData The swap operation's return data.
    function afterSwap(
        address sender,
        PoolKey calldata key,
        IPoolManager.SwapParams calldata params,
        SwapReturnData calldata returnData
    ) external returns (bytes4 selector);
```

Bacon Labs: Fixed in commit 0189567.

Cyfrin: Verified. The reference has been removed.

7.3.3 `BeforeSwapFeeOverride` struct could be used in the `FeeOverride` struct

Description: The `BeforeSwapFeeOverride` and `BeforeSwapPriceOverride` structs defined in `IHooklet` are not currently used:

```
/// @notice Overrides the swap fee of a pool before the swap is executed.
/// Ignored if the pool has an am-AMM manager.
/// @member overridden If true, the swap fee is overridden.
/// @member fee The swap fee to use for the swap. 6 decimals.
struct BeforeSwapFeeOverride {
    bool overridden;
    uint24 fee;
}

/// @notice Overrides the pool's spot price before the swap is executed.
/// @member overridden If true, the pool's spot price is overridden.
/// @member sqrtPriceX96 The spot price to use for the swap. Q96 value.
```

```
struct BeforeSwapPriceOverride {
    bool overridden;
    uint160 sqrtPriceX96;
}
```

The `FeeOverride` struct defined in `FeeOverrideHooklet` could instead be comprised of two `BeforeSwapFeeOverride` members for zero/one:

```
struct FeeOverride {
    bool overrideZeroToOne;
    uint24 feeZeroToOne;
    bool overrideOneToZero;
    uint24 feeOneToZero;
}
```

Recommended Mitigation:

```
    struct FeeOverride {
-       bool overrideZeroToOne;
-       uint24 feeZeroToOne;
-       bool overrideOneToZero;
-       uint24 feeOneToZero;
+       BeforeSwapFeeOverride overrideZeroToOne;
+       BeforeSwapFeeOverride overrideOneToZero;
    }
```

Bacon Labs: Rather than use the struct definitions, we've decided to remove them from the core contracts entirely in commit fc18b04 as they were unused. As such, we will not be implementing the suggested fix.

Cyfrin: Acknowledged. The upstream unused structs have been removed

7.3.4 No need to bound the fee override in `FeeOverrideHooklet::setFeeOverride` as it is clamped in `HookletLib::beforeSwap`

Description: `FeeOverrideHooklet::setFeeOverride` bounds the overridden fees to prevent them from exceeding `SWAP_FEE_BASE`:

```
function setFeeOverride(
    PoolId id,
    bool overrideZeroToOne,
    uint24 feeZeroToOne,
    bool overrideOneToZero,
    uint24 feeOneToZero
) public {
    if (feeZeroToOne >= SWAP_FEE_BASE || feeOneToZero >= SWAP_FEE_BASE) {
        revert FeeOverrideHooklet__InvalidSwapFee();
    }

    ...
}
```

Assuming L-01 is correctly mitigated as suggested, it is not strictly necessary to perform this validation in the hooklet as fees will be clamped by the `HookletLib::beforeSwap` logic.

Recommended Mitigation: Avoid performing the same validation in multiple instances and instead rely on the behavior of core Bunni contracts.

```
    function setFeeOverride(
        PoolId id,
        bool overrideZeroToOne,
        uint24 feeZeroToOne,
        bool overrideOneToZero,
        uint24 feeOneToZero
        ) public {
    -       if (feeZeroToOne >= SWAP_FEE_BASE || feeOneToZero >= SWAP_FEE_BASE) {
    -           revert FeeOverrideHooklet__InvalidSwapFee();
    -       }

            ...
        }
```

Bacon Labs: Acknowledged. We're going to keep this validation vs relying solely on the clamping behavior in the core contracts. Without this validation there is a scenario where a pool curator can set a fee higher than the maximum allowable value, which is then clamped to the maximum value in the core contracts, leading to a swap fee value that differs from the one set by the curator. Keeping this validation removes the potential for confusion and does not otherwise hurt to have it beyond the minor increase in gas costs.

Cyfrin: Acknowledged.