

Coinbase Verified Pools v2

Security Review

Cantina Managed review by:

0xIcingdeath, Lead Security Researcher

RustyRabbit, Security Researcher
0xLuk3, Associate Security Researcher

November 12, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Routers that support price limit in swaps can lead to excessive hook fees . . . 4
  - 3.1.2 Truncation on `unspecifiedDelta` results in user receiving more funds than intended . . . 6
- 3.2 Medium Risk . . . 10
  - 3.2.1 Fee calculations should have explicit rounding upwards instead of being truncated . . . 10
- 3.3 Low Risk . . . 10
  - 3.3.1 `getHookFee()` does not revert for `UNSET` pools . . . 10
  - 3.3.2 SVG validation does not prevent web2 style attacks . . . 11
- 3.4 Informational . . . 11
  - 3.4.1 Chain ID and addresses are unique to base . . . 11
  - 3.4.2 Incorrect batched sender blocking input can be hard to detect . . . 12
  - 3.4.3 `FeeManager` default `swapFee` is not set in the constructor . . . 12
  - 3.4.4 Hardcoded chain ID can cause issue if BASE is hard-forked . . . 12
  - 3.4.5 Improvement to testing . . . 13

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Base is a secure, low-cost, builder-friendly Ethereum L2 built to bring the next billion users onchain.

From Oct 14th to Oct 26th the Cantina team conducted a review of verified-pools on commit hash 36d69b68. The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	1	1	0
Low Risk	2	1	1
Gas Optimizations	0	0	0
Informational	5	3	2
Total	**10**	**7**	**3**

3 Findings

3.1 High Risk

3.1.1 Routers that support price limit in swaps can lead to excessive hook fees

Severity: High Risk

Context: VerifiedPoolsBasicHookV2.sol#L382-L383, VerifiedPoolsBasicHookV2.sol#L401-L402

Summary: The fee calculated in `beforeSwap()` can be to high if the price limit is reached in the swap.

Finding Description: When the amount specified in a swap is the preferred currency it is necessary to calculate and deduct the hook fee in the `beforeswap()` as otherwise it would be used in the swap or the user would not receive the specified amount. The calculation of the fee amount is based on the the specified amount which is presumed to be reached in the swap. This however may not be the case if a price limit is specified in the swap. In such a case the deducted amount will be to high as the actual swapped amount would be less than anticipated.

Note that when a hookDelta is larger than the swapDelta the subtraction in Uniswap's PoolManager's handling of afterSwap this will result in the user being required to pay rather than receive funds.

Impact Explanation: The user will pay a higher fee than expected (even higher than the `HOOK_FEE_CAP`) which may also lead to regulatory issues. Therefore we consider this a high impact.

Likelihood Explanation: The swap router needs to allow the use of a price limit. As swap routers need to be whitelisted and the Uniswap universal router does not support the use of price limit we consider this a medium likelihood.

Proof of Concept: The following diff is a Exactout swap test for the preferred currency with a price limit that shows the excessive fee being applied. The swapped amount is so small compared to the fee that the user is paying a small amount of input currency and practically the full amount of the fee without getting anything in return:

```
  diff --git a/test/PoolSwap.sol b/test/PoolSwap.sol
  index 4e8dee8..14bc93f 100644
-  -- a/test/PoolSwap.sol
+  ++ b/test/PoolSwap.sol
  @@ -69,6 +69,8 @@ contract PoolSwap is PoolTestBase, ReentrancyLock {
           (,, int256 deltaAfter0) = _fetchBalances(data.key.currency0, data.sender, address(this));
           (,, int256 deltaAfter1) = _fetchBalances(data.key.currency1, data.sender, address(this));

+          bool isPriceLimit = bytes32(data.hookData) ==  keccak256(abi.encodePacked("PRICELIMIT"));
+          if (!isPriceLimit){  //bypass the balance checks if hookdata is PRCIELIMIT, this emulate a swap
↪  router that allows a pricelimit to be set and thus returning a swapped amount less than the sepcified amount
           if (data.params.zeroForOne) {
               if (data.params.amountSpecified < 0) {
                   // exact input, 0 for 1
  @@ -106,7 +108,7 @@ contract PoolSwap is PoolTestBase, ReentrancyLock {
                   );
               }
           }
-
+          }
           if (deltaAfter0 < 0) {
               data.key.currency0.settle(manager, data.sender, uint256(-deltaAfter0),
               ↪  data.testSettings.settleUsingBurn);
           }
  diff --git a/test/VerifiedPoolsBasicHookV2/swap.t.sol b/test/VerifiedPoolsBasicHookV2/swap.t.sol
  index 85e1d78..c87c7ea 100644
-  -- a/test/VerifiedPoolsBasicHookV2/swap.t.sol
+  ++ b/test/VerifiedPoolsBasicHookV2/swap.t.sol
  @@ -21,8 +21,11 @@ import {VerifiedPoolsBasicHookV2} from "../../src/VerifiedPoolsBasicHookV2.sol";
   // Test
   import {Currency, CurrencyLibrary, IPoolManager, PoolKey, PoolId} from "../utils/Deployers.sol";
   import {VerifiedPoolsBasicHookV2Test} from "./VerifiedPoolsBasicHookV2.t.sol";
+  import {console} from "forge-std/console.sol";
+  import {StateLibrary} from "@uniswap/v4-core/src/libraries/StateLibrary.sol";

   contract SwapV2Test is VerifiedPoolsBasicHookV2Test {
+      using StateLibrary for IPoolManager;
       modifier whenNon_dynamicFeePool() {
           _;
```

```
      }
  @@ -246,6 +249,20 @@ contract SwapV2Test is VerifiedPoolsBasicHookV2Test {
          _assertSwapExactOut(_dynamicPoolKey, 30 ether, _zeroForOne, hookData, currency0);
      }

+     function test_priceLimit_exactOut()
+         external
+         whenDynamicFeePool
+         whenCallerVerifiedInDynamicPool {
+
+         bytes memory hookData = bytes(abi.encodePacked(keccak256(abi.encodePacked("PRICELIMIT"))));
+         uint8 hookFee=100;  //set 1% hookfee
+         // Test exactOut with input as preferred currency
+         vm.prank(_feeManagerAdmin);
+         _feeManager.setHookFee(_dynamicPoolId, uint128(hookFee), Currency.unwrap(currency1));
+         uint amount = 600_000 ether;
+         _assertSwapExactOut(_dynamicPoolKey, amount, _zeroForOne, hookData, currency1);
+         }
+
      function test_WhenRandomHookDataInDynamicPool(bytes memory hookData)
          external
          whenDynamicFeePool
  @@ -377,7 +394,7 @@ contract SwapV2Test is VerifiedPoolsBasicHookV2Test {
          (_dynamicPoolKey, _dynamicPoolId) = _initializeDynamicFeePool(SQRT_PRICE_1_1);
          _addLiquidity(_dynamicPoolKey);

-         _fundAddress(_swapper, 100 ether);
+         _fundAddress(_swapper, 1_000_000 ether);
          _approveSwap(_swapper, _erc20(currency0));
          _approveSwap(_swapper, _erc20(currency1));
      }
  @@ -423,7 +440,7 @@ contract SwapV2Test is VerifiedPoolsBasicHookV2Test {
              expectedFee = outputDelta * hookFee / (10_000 - hookFee);
          } else {
              // Fee taken from input - for exact input, we know the exact input amount
-             expectedFee = amount * hookFee / 10_000;
+             expectedFee = amount * hookFee / (10_000 + hookFee);
          }

          assertEq(postSwapBalances[0], preSwapBalances[0] - amount);
  @@ -476,10 +493,14 @@ contract SwapV2Test is VerifiedPoolsBasicHookV2Test {
              expectedFee = inputDelta * hookFee / (10_000 + hookFee);
          } else {
              // Fee taken from output - for exact output, we know the exact output amount
-             expectedFee = amount * hookFee / 10_000;
+             expectedFee = amount * hookFee / (10_000 - hookFee);
          }

-         assertEq(postSwapBalances[1], preSwapBalances[1] + amount);
+         if (bytes32(hookData) !=  keccak256(abi.encodePacked("PRICELIMIT"))) {
+             assertEq(int256(postSwapBalances[0]) - int256(preSwapBalances[0]), -int256(amount), "swapper
↪  balance should have change by the specified amount ");
+         } else {
+             // we could have hit the pricelimit so no sense in checking the swapped amount
+         }
          assertGe(preSwapBalances[0], postSwapBalances[0]);

          if (hookFee > 0) {
  @@ -496,10 +517,17 @@ contract SwapV2Test is VerifiedPoolsBasicHookV2Test {
      function _swapWithHookData(PoolKey memory poolKey, int256 amountSpecified, bool zeroForOne, bytes memory
      ↪  hookData)
          internal
      {
+         (uint160 sqrtPriceX96,,,) = IPoolManager(manager).getSlot0(poolKey.toId());
+         uint160 priceLimit ;
+         if (bytes32(hookData) ==  keccak256(abi.encodePacked("PRICELIMIT"))) {
+             priceLimit = zeroForOne ? sqrtPriceX96 - 1e5 : sqrtPriceX96 + 1e5;
+         } else {
+             priceLimit = zeroForOne ? MIN_PRICE_LIMIT : MAX_PRICE_LIMIT;
+             }
          IPoolManager.SwapParams memory params = IPoolManager.SwapParams({
              zeroForOne: zeroForOne,
              amountSpecified: amountSpecified,
-             sqrtPriceLimitX96: zeroForOne ? MIN_PRICE_LIMIT : MAX_PRICE_LIMIT
+             sqrtPriceLimitX96: priceLimit
          });
```

```
        swapRouter.swap(poolKey, params, PoolSwap.TestSettings({takeClaims: false, settleUsingBurn: false}),
        ↪  hookData);
```

Recommendation: Consider refunding the user the overcharged fee amount in the `afterSwap()` where the correct fee amount can be recalculated based on the actual amount swapped. The hook's minted position should also be burnt by the same amount.

Coinbase: Fixed in PR 11.

Cantina Managed: Fixed.

3.1.2 Truncation on `unspecifiedDelta` results in user receiving more funds than intended

Severity: High Risk

Context: VerifiedPoolsBasicHookV2.sol#L450-L475

Description: The `unspecifiedDelta` calculation in cases 3 and 4 end up truncating the total fee amount down. This means the user ends up benefitting from dust from the calculation. The implemented rounding breaks the following invariant:

```
denominator = numerator/feeAmt
feeAmt = numerator / denominator
```

The failure of this invariant holding results in the following assertion statement breaking:

```
assertApproxEqRel(
    postSwapBalances[2],
    preSwapBalances[2] + expectedFee,
    1
);
```

Proof of Concept: The fuzzer found a case in which the post-swap balance is less than pre-swap balance + expectedFee.

```
numerator = 679999967459348139995
feeAmt = 67999996745934813

numerator / feeAmt = denominator
7999996745934139995 / 67999996745934813

num/denom = feeAmt
6799999674593481399995 / 10 000 = should be: 67,999,996,745,934,813.9995
actually;                                                    67,999,996,745,934,813.0
```

The following added assertion depicts the difference between the expected fee amounts on the manager:

```
assertGe(expectedFee, 0);
assertGe(postSwapBalances[2], preSwapBalances[2]);
```

The results are as follows - note that the 2nd index of the balance swap values represent the amounts the managers receive. In the case below, the manager receives 1 wei less than it should:

```
  [0] VM::assertGe(67999996745934814 [6.799e16], 0) [staticcall]
    ← [Return]
  [0] VM::assertGe(67999996745934813 [6.799e16], 0) [staticcall]
    ← [Return]
```

See PR 8 for tests:

Full fuzzer failure with logs:

```
    [196159] PoolSwap::swap(PoolKey({ currency0: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, currency1:
  ↪  0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758, fee: 8388608 [8.388e6], tickSpacing: 60, hooks:
  ↪  0x20712FD390b4C7791A82434c276E933100a8Aaec }), SwapParams({ zeroForOne: true, amountSpecified:
  ↪  -8000000000000000000 [-8e18], sqrtPriceLimitX96: 4295128740 [4.295e9] }), TestSettings({ takeClaims:
  ↪  false, settleUsingBurn: false }), 0x6fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8
  ↪  815bf2e9e67241c8063c3c42109b1c7656163)
```

```
    [191743] PoolManager::unlock(0x0000000000000000000000000000000000000000000000000000000000000020000000
↪ 0000000000000000004f860881b76e70422aea1e53bab22e034ac0124f0000000000000000000000000000000000000000000000000000000000000000
↪ 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪ 000013aa49bac059d709dd0a18d6bb63290076a702d70000000000000000000000096d3f6c20eed2697647f543fe6c08bc2fbf
↪ 3975800000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000000
↪ 0000000000000000000000000003c0000000000000000000000020712fd390b4c7791a82434c276e933100a8aaec000000000
↪ 0000000000000000000000000000000000000000000000000000000000000001ffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪ 90fa4a62c4e0000000000000000000000000000000000000000000000000000000000001000276a400000000000000000000000000
↪ 0000000000000000000000000000000000000018000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪ 346fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656
↪ 16300000000000000000000000)
        [189388] PoolSwap::unlockCallback(0x0000000000000000000000000000000000000000000000000000000000000000
↪ 2000000000000000000000004f860881b76e70422aea1e53bab22e034ac0124f0000000000000000000000000000000000000000000000000000000000000000
↪ 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪ 00000000000013aa49bac059d709dd0a18d6bb63290076a702d70000000000000000000000096d3f6c20eed2697647f543fe6c
↪ 08bc2fbf3975800000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000
↪ 0000000000000000000000000000000003c0000000000000000000000020712fd390b4c7791a82434c276e933100a8aaec0
↪ 0000000000000000000000000000000000000000000000000000000000000001ffffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪ ffffffff90fa4a62c4e0000000000000000000000000000000000000000000000000000000000001000276a400000000000000000000000000
↪ 0000000000000000000000000000000000000018000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪ 00000000346fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c4210
↪ 9b1c765616300000000000000000000000)
            [539] MockERC20::balanceOf(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
                ← [Return] 100000000000000000000 [1e23]
            [2539] MockERC20::balanceOf(PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211])
↪ [staticcall]
                ← [Return] 102000000000000000000000000 [1.02e26]
            [369]
↪ PoolManager::exttload(0xba117c8eaa889ac864808748724e58c8c5c458c62066fabf4753d23aa18a1e9c) [staticcall]
                ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000000
            [539] MockERC20::balanceOf(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
                ← [Return] 100000000000000000000 [1e23]
            [2539] MockERC20::balanceOf(PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211])
↪ [staticcall]
                ← [Return] 102000000000000000000000000 [1.02e26]
            [369]
↪ PoolManager::exttload(0x1624e3b16164c9d99599fc75f399f4716ecd1c53126398141b37e22170ae7a11) [staticcall]
                ← [Return] 0x0000000000000000000000000000000000000000000000000000000000000000
            [140682] PoolManager::swap(PoolKey({ currency0: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7,
↪ currency1: 0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758, fee: 8388608 [8.388e6], tickSpacing: 60, hooks:
↪ 0x20712FD390b4C7791A82434c276E933100a8Aaec }), SwapParams({ zeroForOne: true, amountSpecified:
↪ -8000000000000000000 [-8e18], sqrtPriceLimitX96: 4295128740 [4.295e9] }), 0x6fb9968386030136ae10a764c5c
↪ b794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656163)
                [66080] VerifiedPoolsBasicHookV2::beforeSwap(PoolSwap:
↪ [0x03A6a84cD762D9707A21605b548aaaB891562aAb], PoolKey({ currency0:
↪ 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, currency1: 0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758, fee:
↪ 8388608 [8.388e6], tickSpacing: 60, hooks: 0x20712FD390b4C7791A82434c276E933100a8Aaec }), SwapParams({
↪ zeroForOne: true, amountSpecified: -8000000000000000000 [-8e18], sqrtPriceLimitX96: 4295128740
↪ [4.295e9] }), 0x6fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c
↪ 3c42109b1c7656163)
                    [4706] VerifiedPoolsPolicyRegistryV2::getBeforeSwapPolicyForPool(0x3bdf5b7b04fa4dabc372
↪ b17599233f6f6eb9bdcba0d036514f7733260b248355) [staticcall]
                        ← [Return] BasicPolicy: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a]
                    [356] PoolSwap::msgSender() [staticcall]
                        ← [Return] Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]
                    [32852] BasicPolicy::verify(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F], 0x6f
↪ b9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656163)
↪ [staticcall]
                        [612] MockAttestationIndexer::getAttestationUid(Swapper:
↪ [0x4f860881b76E70422aEA1e53BAb22E034AC0124F],
↪ 0x48bb3eba1e4c1fa1b7f92843bd631d8145aa8689c951adcf28ddc8b42b6a6c99) [staticcall]
                            ← [Return] 0xf408eb3bdc532bb89794010ddd7f3f62ff4b19a8694230ec8f5bdb283a5b278b
                        [3205]
↪ EAS::getAttestation(0xf408eb3bdc532bb89794010ddd7f3f62ff4b19a8694230ec8f5bdb283a5b278b) [staticcall]
                            ← [Return] Attestation({ uid:
↪ 0xf408eb3bdc532bb89794010ddd7f3f62ff4b19a8694230ec8f5bdb283a5b278b, schema:
↪ 0x48bb3eba1e4c1fa1b7f92843bd631d8145aa8689c951adcf28ddc8b42b6a6c99, time: 1, expirationTime: 0,
↪ revocationTime: 0, refUID: 0x0000000000000000000000000000000000000000000000000000000000000000,
↪ recipient: 0x4f860881b76E70422aEA1e53BAb22E034AC0124F, attester:
↪ 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496, revocable: true, data:
↪ 0x0000000000000000000000000000000000000000000000000000000000000001 })
                        [612] MockAttestationIndexer::getAttestationUid(Swapper:
↪ [0x4f860881b76E70422aEA1e53BAb22E034AC0124F],
↪ 0xc93a4feaf986b4cba6f3038e313a22219522292a15219ee1d465f918ecee7d2a) [staticcall]
                            ← [Return] 0xdc965bdc50b2e338ee80c0e0ed51f706bc6bac501ee2124e14f10f27cff4007a
```

```
                    [3651]
↪ EAS::getAttestation(0xdc965bdc50b2e338ee80c0e0ed51f706bc6bac501ee2124e14f10f27cff4007a) [staticcall]
                        ← [Return] Attestation({ uid:
↪ 0xdc965bdc50b2e338ee80c0e0ed51f706bc6bac501ee2124e14f10f27cff4007a, schema:
↪ 0xc93a4feaf986b4cba6f3038e313a22219522292a15219ee1d465f918ecee7d2a, time: 1, expirationTime: 0,
↪ revocationTime: 0, refUID: 0x0000000000000000000000000000000000000000000000000000000000000000,
↪ recipient: 0x4f860881b76E70422aEA1e53BAb22E034AC0124F, attester:
↪ 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496, revocable: true, data:
↪ 0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000
↪ 0000000000000000000000000255300000000000000000000000000000000000000000000000000000000000000000 })
                    [2546] MockSanctionsList::isSanctioned(Swapper:
↪ [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
                        ← [Return] false
                    ← [Return] true
                [3803] VerifiedPoolsFeeManagerV2::getSwapFee(0x3bdf5b7b04fa4dabc372b17599233f6f6eb9bdcb
↪ a0d036514f7733260b248355, PoolSwap: [0x03A6a84cD762D9707A21605b548aaaB891562aAb], SwapParams({
↪ zeroForOne: true, amountSpecified: -8000000000000000000 [-8e18], sqrtPriceLimitX96: 4295128740
↪ [4.295e9] }), 0x6fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c
↪ 3c42109b1c7656163) [staticcall]
                    ← [Return] 0
                [1609] VerifiedPoolsFeeManagerV2::getHookFee(0x3bdf5b7b04fa4dabc372b17599233f6f6eb9bdcb
↪ a0d036514f7733260b248355, PoolSwap: [0x03A6a84cD762D9707A21605b548aaaB891562aAb],
↪ 0x6fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656
↪ 163) [staticcall]
                    ← [Return] 85
                [1256] VerifiedPoolsFeeManagerV2::getPreferredHookFeeCurrency(0x3bdf5b7b04fa4dabc372b17
↪ 599233f6f6eb9bdcba0d036514f7733260b248355) [staticcall]
                    ← [Return] MockERC20: [0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758]
                ← [Return] 0x575e24b4, 0, 4194304 [4.194e6]
            emit Swap(id: 0x3bdf5b7b04fa4dabc372b17599233f6f6eb9bdcba0d036514f7733260b248355, sender:
↪ PoolSwap: [0x03A6a84cD762D9707A21605b548aaaB891562aAb], amount0: -8000000000000000000 [-8e18], amount1:
↪ 7999999617168801647 [7.999e18], sqrtPriceX96: 792281587228877877667368654 52 [7.922e28], liquidity:
↪ 167175499835819766909277567 [1.671e26], tick: -1, fee: 0)
            [33012] VerifiedPoolsBasicHookV2::afterSwap(PoolSwap:
↪ [0x03A6a84cD762D9707A21605b548aaaB891562aAb], PoolKey({ currency0:
↪ 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, currency1: 0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758, fee:
↪ 8388608 [8.388e6], tickSpacing: 60, hooks: 0x20712FD390b4C7791A82434c276E933100a8Aaec }), SwapParams({
↪ zeroForOne: true, amountSpecified: -8000000000000000000 [-8e18], sqrtPriceLimitX96: 4295128740
↪ [4.295e9] }), -272225893536750770770699685945414569164000000382831198353 [-2.722e57], 0x6fb99683860301
↪ 36ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656163)
                [1609] VerifiedPoolsFeeManagerV2::getHookFee(0x3bdf5b7b04fa4dabc372b17599233f6f6eb9bdcb
↪ a0d036514f7733260b248355, PoolSwap: [0x03A6a84cD762D9707A21605b548aaaB891562aAb],
↪ 0x6fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656
↪ 163) [staticcall]
                    ← [Return] 85
                [1256] VerifiedPoolsFeeManagerV2::getPreferredHookFeeCurrency(0x3bdf5b7b04fa4dabc372b17
↪ 599233f6f6eb9bdcba0d036514f7733260b248355) [staticcall]
                    ← [Return] MockERC20: [0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758]
                [24525] PoolManager::mint(VerifiedPoolsBasicHookV2:
↪ [0x20712FD390b4C7791A82434c276E933100a8Aaec], 861075568517424834823154086803965909965347198808
↪ [8.61e47], 67999996745934813 [6.799e16])
                    emit Transfer(caller: VerifiedPoolsBasicHookV2:
↪ [0x20712FD390b4C7791A82434c276E933100a8Aaec], from: 0x0000000000000000000000000000000000000000, to:
↪ VerifiedPoolsBasicHookV2: [0x20712FD390b4C7791A82434c276E933100a8Aaec], id:
↪ 861075568517424834823154086803965909965347198808 [8.61e47], amount: 67999996745934813 [6.799e16])
                    ← [Stop]
                ← [Return] 0xb47b2fb1, 67999996745934813 [6.799e16]
            ← [Return] -272225893536750770770699685945414569164006800037957713316 6 [-2.722e57]
        [539] MockERC20::balanceOf(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
            ← [Return] 100000000000000000000000 [1e23]
        [539] MockERC20::balanceOf(PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211])
↪ [staticcall]
            ← [Return] 102000000000000000000000000 [1.02e26]
        [369]
↪ PoolManager::exttload(0xba117c8eaa889ac864808748724e58c8c5c458c62066fabf4753d23aa18a1e9c) [staticcall]
            ← [Return] 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff90fa4a62c4e00000
        [539] MockERC20::balanceOf(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
            ← [Return] 100000000000000000000000 [1e23]
        [539] MockERC20::balanceOf(PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211])
↪ [staticcall]
            ← [Return] 102000000000000000000000000 [1.02e26]
        [369]
↪ PoolManager::exttload(0x1624e3b16164c9d99599fc75f399f4716ecd1c53126398141b37e22170ae7a11) [staticcall]
            ← [Return] 0x0000000000000000000000000000000000000000000000000006e141fa115d69792
        [1678] PoolManager::sync(MockERC20: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7])
```

```
                [539] MockERC20::balanceOf(PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211])
↪  [staticcall]
                    ← [Return] 1020000000000000000000000 [1.02e26]
                ← [Stop]
            [10977] MockERC20::transferFrom(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F],
↪  PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211], 8000000000000000000 [8e18])
                emit Transfer(from: Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F], to:
↪  PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211], value: 8000000000000000000 [8e18])
                ← [Return] true
            [2949] PoolManager::settle()
                [539] MockERC20::balanceOf(PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211])
↪  [staticcall]
                    ← [Return] 1020000080000000000000000 [1.02e26]
                ← [Return] 8000000000000000000 [8e18]
            [10658] PoolManager::take(MockERC20: [0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758], Swapper:
↪  [0x4f860881b76E70422aEA1e53BAb22E034AC0124F], 7931999620422866834 [7.931e18])
                [8580] MockERC20::transfer(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F],
↪  7931999620422866834 [7.931e18])
                    emit Transfer(from: PoolManager: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211], to:
↪  Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F], value: 7931999620422866834 [7.931e18])
                    ← [Return] true
                ← [Stop]
            ← [Return] 0xffffffffffffffff90fa4a62c4e0000000000000000000006e141fa115d69792
        ← [Return] 0xffffffffffffffff90fa4a62c4e0000000000000000000006e141fa115d69792
    ← [Return] -272225893536750770770699685945414569164006800037957713316 6 [-2.722e57]
  [0] VM::getRecordedLogs()
    ← [Return] [([0x40e9cecb9f5f1f1c5b9c97dec2917b7ee92e57ba5563708daca94dd84ad7112f,
↪  0x3bdf5b7b04fa4dabc372b17599233f6f6eb9bdcba0d036514f7733260b248355,
↪  0x0000000000000000000000003a6a84cd762d9707a21605b548aaab891562aab],
↪  0xffffffffffffffffffffffffffffffffffffffffffffffff90fa4a62c4e0000000000000000000000000000000000000000
↪  000000000006f05b544189b7f6f0000000000000000000000000000000000000000ffffff327814ab81a311c8ac000000000000
↪  0000000000000000000000000008a48ca0163071ccd2c957ffffffffffffffffffffffffffffffffffffffffffffffffffffff
↪  fffffffffffff0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000,
↪  0x1d1499e622D69689cdf9004d05Ec547d650Ff211),
↪  ([0x1b3d7edb2e9c0b0e7c525b20aaaef0f5940d2ed71663c7d39266ecafac728859,
↪  0x0000000000000000000000000000000000000000000000000000000000000000,
↪  0x0000000000000000000000020712fd390b4c7791a82434c276e933100a8aaec,
↪  0x0000000000000000000000096d3f6c20eed2697647f543fe6c08bc2fbf39758], 0x0000000000000000000000020712fd3
↪  90b4c7791a82434c276e933100a8aaec0000000000000000000000000000000000000000000000000000000000000000f195a302c4e7dd,
↪  0x1d1499e622D69689cdf9004d05Ec547d650Ff211),
↪  ([0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef,
↪  0x0000000000000000000000004f860881b76e70422aea1e53bab22e034ac0124f,
↪  0x0000000000000000000000001d1499e622d69689cdf9004d05ec547d650ff211],
↪  0x0000000000000000000000000000000000000000000000000000006f05b59d3b200000,
↪  0x13aa49bAc059d709dd0a18D6bb63290076a702D7),
↪  ([0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef,
↪  0x0000000000000000000000001d1499e622d69689cdf9004d05ec547d650ff211,
↪  0x0000000000000000000000004f860881b76e70422aea1e53bab22e034ac0124f],
↪  0x0000000000000000000000000000000000000000000000006e141fa115d69792,
↪  0x96d3F6c20EEd2697647F543fE6C08bC2Fbf39758)]
  [1609] VerifiedPoolsFeeManagerV2::getHookFee(0x3bdf5b7b04fa4dabc372b17599233f6f6eb9bdcba0d036514f7733260b
↪  248355, 0x0000000000000000000000000000000000000000, 0x) [staticcall]
    ← [Return] 85
  [0] VM::assertEq(4, 4) [staticcall]
    ← [Return]
  [539] MockERC20::balanceOf(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
    ← [Return] 9999200000000000000000 [9.999e22]
  [539] MockERC20::balanceOf(Swapper: [0x4f860881b76E70422aEA1e53BAb22E034AC0124F]) [staticcall]
    ← [Return] 1000079319996204228668 34 [1e23]
  [639] PoolManager::balanceOf(VerifiedPoolsBasicHookV2: [0x20712FD390b4C7791A82434c276E933100a8Aaec],
↪  861075568517424834823154086803965909965347198808 [8.61e47]) [staticcall]
    ← [Return] 67999996745934813 [6.799e16]
  [0] VM::assertGe(67999996745934814 [6.799e16], 0) [staticcall]
    ← [Return]
  [0] VM::assertGe(67999996745934813 [6.799e16], 0) [staticcall]
    ← [Return]
  [0] VM::assertEq(9999200000000000000000 [9.999e22], 9999200000000000000000 [9.999e22]) [staticcall]
    ← [Return]
  [0] VM::assertGe(1000079319996204228668 34 [1e23], 100000000000000000000 [1e23]) [staticcall]
    ← [Return]
  [0] VM::assertApproxEqRel(67999996745934813 [6.799e16], 67999996745934814 [6.799e16], 10) [staticcall]
    ← [Revert] assertion failed: 67999996745934813 !~= 67999996745934814 (max delta: 0.0000000000000010%,
↪  real delta: 0.0000000000000014%)
  ← [Revert] assertion failed: 67999996745934813 !~= 67999996745934814 (max delta: 0.0000000000000010%,
↪  real delta: 0.0000000000000014%)
```

```
Backtrace:
  at VM.assertApproxEqRel
  at SwapV2Test.testFuzz_WhenNonZeroHookFeeInDynamicPool

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 430.57ms (5.33ms CPU time)

Ran 1 test suite in 445.77ms (430.57ms CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/VerifiedPoolsBasicHookV2/swap.t.sol:SwapV2Test
[FAIL: assertion failed: 67999996745934813 !~= 67999996745934814 (max delta: 0.0000000000000010%, real delta:
↪  0.0000000000000014%); counterexample:
↪  calldata=0x8f218a11000000000000000000000000000000000000000000000000000000000000005500000000000000000000000000
↪  0000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000000060000
↪  0000000000000000000000000000000000000000000000000000000000000346fb9968386030136ae10a764c5cb794df501d15f5f87a2
↪  b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c765616300000000000000000000000 args=[85, 8,
↪  0x6fb9968386030136ae10a764c5cb794df501d15f5f87a2b4422f26d23774338d0c8815bf2e9e67241c8063c3c42109b1c7656163]
↪  ] testFuzz_WhenNonZeroHookFeeInDynamicPool(uint8,uint128,bytes) (runs: 0,  : 0, ~: 0)

Encountered a total of 1 failing tests, 0 tests succeeded
```

Recommendation: Round the calculated fee up, to ensure that the pool manager receives any dust as a result of calculations.

Coinbase: Tests have been updated in PR 11.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Fee calculations should have explicit rounding upwards instead of being truncated

Severity: Medium Risk

Context: VerifiedPoolsBasicHookV2.sol#L905

Finding Description: The `beforeSwap` and `afterSwap` functions work on `uint128`/`int128` values. However, the beforeSwap function uses two helper functions - `_calculateScaledFeeForExactOutput` and `_calculateScaledFeeForExactInput` in cases 1 and 2, respectively.

In these two helper functions, the values are scaled *upwards* to `uint256` values, the numerator and denominator are used to calculate the fee percentage. Most importantly, the value is then *downcasted* to an `int128`. This means the fee is consistently going to be truncated downwards - the protocol will effectively receive less funds than intended.

Recommendation: Explicitly round the `numerator/denominator` calculation upwards, such that the contract takes the extra remainder from precision errors. Conduct additional analysis on whether the `uint256` casting is necessary in the helper function -- if not, consider calculating the `scaledFee` in a `uint128`.

Coinbase: Fixed in PR 11.

Cantina Managed: Fixed.

3.3 Low Risk

3.3.1 `getHookFee()` does not revert for `UNSET` pools

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In `VerifiedPoolsFeeManagerV2.sol`, the default fee logic does not meet intended design requirements. In `getHookFee()` anything different than `FeeState.EXPLICIT` falls back to return `feeConfigurations[DEFAULT_POOL].hookFee`. However the intended design is to that both hook fees and preferred currencies must be explicitly configured per pool and never inherited from the `DEFAULT_POOL`.

```
function getHookFee(PoolId poolId, address, /*sender*/ bytes calldata /*hookData*/ )
    external
    view
    override
    returns (uint128)
```

```
{
    FeeConfiguration memory config = feeConfigurations[poolId];
    if (config.hookFeeState == FeeState.EXPLICIT) {
        return config.hookFee;
    } else {
        return feeConfigurations[DEFAULT_POOL].hookFee;
    }
}
```

Recommendation: Update the `getHookFee()` to revert for `UNSET` pools, which is consistent with the behavior of `getPreferredHookFeeCurrency()`.

Coinbase: Fixed in PR 13.

Cantina Managed: Fix verified.

3.3.2 SVG validation does not prevent web2 style attacks

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In the libraries related to processing of SVG used in the protocol NFTs, which are `SVG.sol`, `SVGV2.sol` and `Descriptor.sol`, there is a countermeasure preventing usage of special characters in the SVG images. The validation of SVG images is needed since this type of images consist of XML syntax and may contain Javascript which in turn may be abused to attack users with e.g. cross-site scripting or similar web2 attacks. This validation is performed by function `isSpecialCharacter`, part of `escapeSpecialCharacters`. However, the validation does not protect from web2 attacks described above.

This is because function `escapeSpecialCharacters` which is used as the sanitization routine prepends certain characters with a backslash.

```
// ...
function constructTokenURI(ConstructTokenURIParams memory params) internal pure returns (string memory) {
    string memory name = generateName(params, feeToPercentString(params.fee));
    string memory descriptionPartOne = generateDescriptionPartOne(
        escapeSpecialCharacters(params.quoteCurrencySymbol),
        escapeSpecialCharacters(params.baseCurrencySymbol),
        addressToString(params.poolManager)
    );
    // ...
```

Function `escapeSpecialCharacters` behaves accordingly to the comment and `Escapes special characters in a string if they are present`. The special characters are detected by following routine. Additionally, those characters are not harmful in most cases.

```
function isSpecialCharacter(bytes1 b) internal pure returns (bool) {
    return b == '"' || b == "\u000c" || b == "\n" || b == "\r" || b == "\t";
}
```

For instance, input such as `</text><script>alert('x')</script><text>` is allowed and creates an SVG with embedded script. While the affected function are read only and still would require a token with crafted name to create a dangerous SVG, in case of reuse of the validation routine.

Recommendation: Better solution to sanitize potential dangerous inputs would be to allow only alphanumeric characters or implement sanitization similar to e.g. sanitize-html.

Coinbase: Acknowledged. Will not fix as we would like to maintain consistency with uniswap's implementation.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Chain ID and addresses are unique to base

Severity: Informational

Context: VerifiedPoolsPositionDescriptorV2.sol#L113

Summary: If the `VerifiedPools` logic is intended to be deployed on alternative chains, the code in the PositionDescriptor need to change.

Finding Description: The `currencyRatioPriority` function checks against pre-determined addresses for tokens and specific to the chain ID. If the system is to be deployed on another chain, the ratio priority will merely return 0, as the new chain ID should be different. The addresses used to compare against should also be changed, if the blockchain does not guarantee unique deployment addresses for these tokens.

```
// base addresses
address private constant BASE_USDC = 0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913;
address private constant BASE_USDT = 0xfde4C96c8593536E31F229EA8f37b2ADa2699bb2;
address private constant BASE_USDS = 0x820C137fa70C8691f0e44Dc420a5e53c168921Dc;
address private constant BASE_DAI = 0x50c5725949A6F0c72E6C4a641F24049A917DB0Cb;
address private constant BASE_CBETH = 0x2Ae3F1Ec7F1F5012CFEab0185bfc7aa3cf0DEc22;
address private constant BASE_CBBTC = 0xcbB7C0000aB88B473b1f5aFd9ef808440eed33Bf;

IPoolManager public immutable poolManager;
address public constant wrappedNative = 0x4200000000000000000000000000000000000006;
```

Recommendation: Add a comment in the code to specify that the code needs to be updated if deployed on a different blockchain. For the new blockchain, ensure the addresses for the tokens are well-formed and correct, and that the behaviour matches what is expected of the function.

Coinbase: Documentation added in PR 14.

Cantina Managed: Fixed.

3.4.2 Incorrect batched sender blocking input can be hard to detect

Severity: Informational

Context: TransferPolicyV2.sol#L139-L147

Description: When batching senders to be blocked any sender that was already blocked is skipped and an event is logged but no information is directly returned to the caller. Offchain monitoring for (non-existing) events must be done to detect any incorrectly provided accounts.

Recommendation: Consider returning an array of the address that were either successfully or unsuccessfully so the caller can more easily take appropriate action.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 `FeeManager` default `swapFee` is not set in the constructor

Severity: Informational

Context: VerifiedPoolsFeeManagerV2.sol#L52-L63

Description: The `FeeManager`'s `getSwapFee()` returns the `DEFAULT_POOL`'s `swapFee`. However this is not set in the constructor and thus will return 0 by default.

Recommendation: Consider setting the `DEFAULT_POOL`'s `swapFee` in the constructor.

Coinbase: Fixed in PR 15.

Cantina Managed: Fixed.

3.4.4 Hardcoded chain ID can cause issue if BASE is hard-forked

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The codebase relies on a hard-coded chain ID of `8453`. If BASE is hardforked, the forked chain will have the *same* chain ID. This means the `currencyRatioPriority` function will end up returning an incorrect value. As a result, hard-coded chain ID's are not recommended.

Recommendation: Avoid the use of hard-coded chain ID's, and use `block.chainID` instead.

Coinbase: Acknowledged, will not fix.

Cantina Managed: Acknowledged.

3.4.5 Improvement to testing

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: It's likely there are additional issues related to rounding within the fee calculation, especially across different values. This is not being found within the current test suite,d ue to the use of static constant values. The following changes were implemented (as part of issue 8).

Recommendation: We recommend implementing the following adjustments to the test suite:

1. Refactoring `test_` functions to `testFuzz_`.
2. Convert amounts to parameterized values (taking `amount` as input).
3. Implement `vm.assume(amount>0 && amount amount<###)`, which limits the `amount` to a smaller value.
4. Convert assertions to use `amount` instead of constant values.
5. Adjust `assertApproxEq` for the specific token amount.

Coinbase: Fixed in PR 11.

Cantina Managed: Implemented. We recommend continuing to extend to more complex fuzzers too.