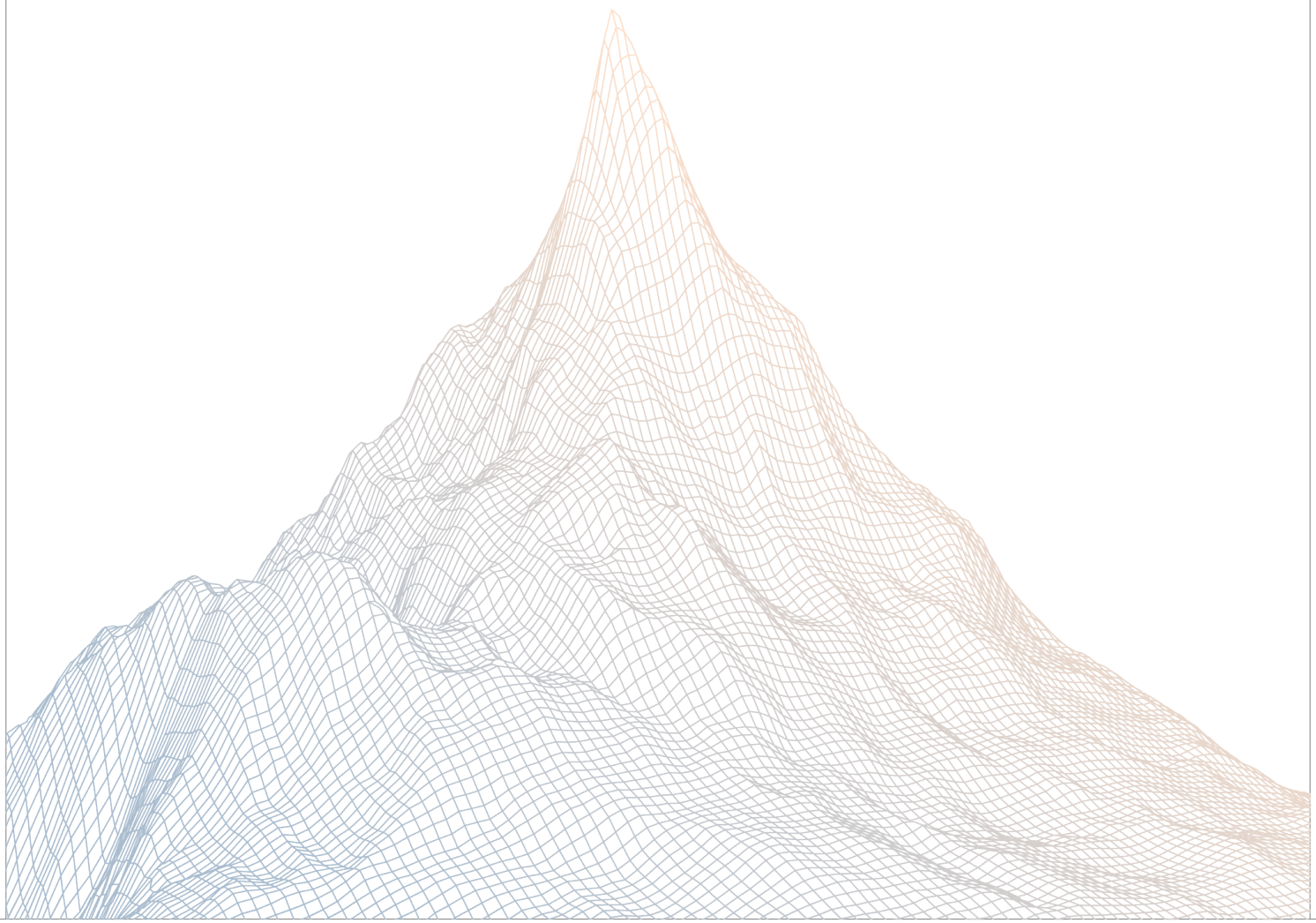
Zenith

Sorella

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: September 12th to September 25th, 2025

AUDITED BY: hickuphh3
zzykxx

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About Sorella 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 6

4.1 High Risk 7

4.2 Medium Risk 15

4.3 Low Risk 21

4.4 Informational 28

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Sorella

Sorella Labs is a technology company based in New York. Using game theory and mechanism design it creates novel solutions to make on-chain finance more efficient.

2.2 Scope

The engagement involved a review of the following targets:

Target	l2-angstrom
Repository	https://github.com/SorellaLabs/l2-angstrom
Commit Hash	`e880ec452106629ca48f9330a7add34bcd9c38b1`
Files	`src/*`

2.3 Audit Timeline

September 12, 2025	Audit start
September 25, 2025	Audit end
September 30, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	2
Medium Risk	2
Low Risk	4
Informational	6
Total Issues	**14**

3

Findings Summary

ID	Description	Status
H-1	zeroForOne current tick transition is incorrect	Resolved
H-2	AFTER_SWAP_RETURNS_DELTA_FLAG isn't turned on	Resolved
M-1	beforeInitialization() checks are bypassed due to noSelfCall modifier	Resolved
M-2	Liquidity ranges deltas are not rounded in the right direction	Resolved
L-1	Fee mutators may cause total swap / tax fees to exceed 100%	Resolved
L-2	Rewards are lost on single-tick swaps with no active liquidity	Acknowledged
L-3	pstarX96 rounds down to zero on low ticks ranges leading to reverts for division by zero	Resolved
L-4	JIT tax can be avoided by first doing a small swap	Resolved
I-1	Native ID conversion to currency can be replaced with NATIVE_CURRENCY constant	Resolved
I-2	Redundant checks on params.liquidityDelta	Acknowledged
I-3	_oneForZeroCreditRewards() calculates rewards of ranges with 0 liquidity	Resolved
I-4	The updateAfterLiquidityAdd() function doesn't initialize the reward accumulator of uninitialized ticks	Resolved
I-5	Indirectly claiming rewards is subject to MEV tax	Resolved
I-6	Redundancies	Resolved

4

Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] `zeroForOne` current tick transition is incorrect

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

CompensationPriceFinder.sol#L67

Description:

When a position gets created, whose `upperTick` matches the pool's current tick, its liquidity isn't added in the pool liquidity yet, because `upperTick` is exclusive and will get added when the price moves down. However, the liquidity addition is not done for such positions, because the `TickIteratorDown` initialisation calls `reset()` which contains a call to `_advanceToNextDown()`, pushing past the current tick. This results in a liquidity subtraction overflow in `zeroForOne` because it will try to subtract liquidity that wasn't added prior.

POC:

```
function test_simpleSwapDown() public {
    PoolKey memory key = initializePool(address(token), 1, 1000);
    addLiquidity(key, 900, 1002, 1234e5);
    addLiquidity(key, 800, 1000, 5678e5);
    (,,,,, uint128 liquidity) = manager.getPool(key.toId());
    console.log("pool liquidity", liquidity);
    setPriorityFee(1 gwei);
    router.swap(key, true, 1e10, TickMath.MIN_SQRT_PRICE + 1);
}
```

In this POC, at the current tick of 1000, the 2nd's position liquidity of `5678e5` needs to be added when transitioning downwards, but isn't. Hence, when the current tick reaches 800, the current liquidity is 0, but `liquidityNet` is `5678e5`.

Recommendations:

Start iterating from a tick higher in `reset()`.

```
diff --git a/lib/v4-periphery b/lib/v4-periphery
-- a/lib/v4-periphery
++ b/lib/v4-periphery
@@ -1 +1 @@
Subproject commit 60cd93803ac2b7fa65fd6cd351fd5fd4cc8c9db5
Subproject commit 60cd93803ac2b7fa65fd6cd351fd5fd4cc8c9db5-dirty
diff --git a/src/hook-config.sol b/src/hook-config.sol
index 1f5313a..e201d45 100644
-- a/src/hook-config.sol
++ b/src/hook-config.sol
@@ -17,4 +17,5 @@ function getRequiredHookPermissions()
    pure returns (Hooks.Permissions memory per
     permissions.beforeSwap = true; // To tax ToB
     permissions.afterSwap = true; // Also to tax with ToB (after swap
    contains reward dist. calculations)
     permissions.beforeSwapReturnDelta = true; // To charge the ToB MEV tax.

    permissions.afterSwapReturnDelta = true; // To charge the protocol swap
        fee
 }
diff --git a/src/libraries/TickIterator.sol b/src/libraries/TickIterator.sol
index b028c0e..68d5f2c 100644
-- a/src/libraries/TickIterator.sol
++ b/src/libraries/TickIterator.sol
@@ -139,6 +139,7 @@ library TickIteratorLib {
     }

     function reset(TickIteratorDown memory self, int24 startTick)
    internal view {
        ++startTick;
         if (!(self.endTick <= startTick)) revert InvalidRange();
         self.currentTick = startTick;

diff --git a/test/AngstromL2.t.sol b/test/AngstromL2.t.sol
index 6afb72a..038f54d 100644
-- a/test/AngstromL2.t.sol
++ b/test/AngstromL2.t.sol
@@ -1048,4 +1048,12 @@ contract AngstromL2Test is BaseTest {
         vm.expectRevert(AngstromL2.IncompatiblePoolConfiguration.selector);
```

```
        angstrom.initializeNewPool(key, INIT_SQRT_PRICE, 0, 0);
    }

    function test_simpleSwapDown() public {
        PoolKey memory key = initializePool(address(token), 1, 1000);
        addLiquidity(key, 900, 1002, 1234e5);
        addLiquidity(key, 800, 1000, 5678e5);
        setPriorityFee(1 gwei);
        router.swap(key, true, 1e10, TickMath.MIN_SQRT_PRICE + 1);
    }
}
diff --git a/test/libraries/TickIterator.t.sol
    b/test/libraries/TickIterator.t.sol
index e08ce37..87f2c4c 100644
-- a/test/libraries/TickIterator.t.sol
++ b/test/libraries/TickIterator.t.sol
@@ -295,13 +295,16 @@ contract TickIteratorTest is BaseTest {

        // Should iterate through ticks in reverse (excluding boundaries)
        assertTrue(iter.hasNext(), "Should have first tick");
        assertEq(iter.getNext(), 50, "First tick should be 50");
        assertEq(iter.getNext(), 100, "First tick should be 100");

        assertTrue(iter.hasNext(), "Should have second tick");
        assertEq(iter.getNext(), 0, "Second tick should be 0");
        assertEq(iter.getNext(), 50, "Second tick should be 50");

        assertTrue(iter.hasNext(), "Should have third tick");
        assertEq(iter.getNext(), -50, "Third tick should be -50");
        assertEq(iter.getNext(), 0, "Third tick should be 0");

        assertTrue(iter.hasNext(), "Should have fourth tick");
        assertEq(iter.getNext(), -50, "Fourth tick should be -50");

        assertFalse(iter.hasNext(), "Should have no more ticks");
    }
@@ -313,12 +316,15 @@ contract TickIteratorTest is BaseTest {
        addLiquidityAtTicks(0, 100);
        addLiquidityAtTicks(100, 200);

        // With exclusive bounds (100, -100)
        // With exclusive bounds (-100, 100]
```

```
         TickIteratorDown memory iter =
             TickIteratorLib.initDown(manager, pid, TICK_SPACING, 100, -100);

         assertTrue(iter.hasNext());
         assertEq(iter.getNext(), 0, "Should exclude start boundary 100");
         assertEq(iter.getNext(), 100, "Should include start boundary 100");

         assertTrue(iter.hasNext());
         assertEq(iter.getNext(), 0);

         assertFalse(iter.hasNext(), "Should exclude end boundary -100");
     }
@@ -381,13 +387,15 @@ contract TickIteratorTest is BaseTest {
     function test_iterateDown_singleTick() public {
         addLiquidityAtTicks(40, 60);

         // With exclusive bounds (60, 40), both boundaries are excluded
         // With exclusive bounds (60, 40], only includes 40
         TickIteratorDown memory iter =
            TickIteratorLib.initDown(manager, pid, TICK_SPACING, 60, 40);

         assertFalse(iter.hasNext(), "Should have no ticks with exclusive
             boundaries");
         assertTrue(iter.hasNext(), "Should have just the 1st tick");
         assertEq(iter.getNext(), 60, "First tick should be 60");
         assertFalse(iter.hasNext(), "Should have no more ticks");

         // To get the boundary ticks, need to expand range
         TickIteratorDown memory iter2 =
             TickIteratorLib.initDown(manager, pid, TICK_SPACING, 70, 30);
         TickIteratorDown memory iter2 =
             TickIteratorLib.initDown(manager, pid, TICK_SPACING, 60, 30);

         assertTrue(iter2.hasNext());
         assertEq(iter2.getNext(), 60, "First tick should be 60");
@@ -583,10 +591,13 @@ contract TickIteratorTest is BaseTest {
         addLiquidityAtTicks(-100, 0);
         addLiquidityAtTicks(0, 100);


     // With exclusive bounds (100, -100), should not include 100 or -100
         // With bounds (-100, 100], includes 100 but not -100
```

```
        TickIteratorDown memory iter =
            TickIteratorLib.initDown(manager, pid, TICK_SPACING, 100, -100);

        assertTrue(iter.hasNext());
        assertEq(iter.getNext(), 100, "Should get tick 100");

        assertTrue(iter.hasNext());
        assertEq(iter.getNext(), 0, "Should only get tick 0");
```

An alternative fix is to check if the start tick is initialized, and return if it is.

```
diff --git a/src/libraries/TickIterator.sol b/src/libraries/TickIterator.sol
index b028c0e..401bff2 100644
-- a/src/libraries/TickIterator.sol
++ b/src/libraries/TickIterator.sol
@@ -144,9 +144,14 @@ library TickIteratorLib {

        if (startTick == self.endTick) return;

        (int16 wordPos,) = TickLib.position(startTick.compress(self.
            tickSpacing));

    (int16 wordPos, uint8 bitPos) = TickLib.position(startTick.compress(
            self.tickSpacing));
        self.currentWord = self.manager.getPoolBitmapInfo(self.poolId,
    wordPos);

        bool initialized;
        (initialized, bitPos) = self.currentWord.nextBitPosLte(bitPos);

    self.currentTick = TickLib.toTick(wordPos, bitPos, self.tickSpacing);
        if (initialized) return;

        _advanceToNextDown(self);
     }
```

Sorella: Resolved with @c01b6c7...

Zenith: Verified.

[H-2] AFTER_SWAP_RETURNS_DELTA_FLAG isn't turned on

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- hook-config.sol#L17-L19, AngstromL2.sol#L284

Description:

The `afterSwap()` function returns the protocol swap fee to be charged, but the `afterSwapReturnDelta` flag isn't set to `true` in the config, so the `PoolManager` will not parse the return value (it's effectively 0).

POC:

```
diff --git a/test/AngstromL2.t.sol b/test/AngstromL2.t.sol
index 6afb72a..af46f7a 100644
-- a/test/AngstromL2.t.sol
++ b/test/AngstromL2.t.sol
@@ -67,6 +67,7 @@ contract AngstromL2Test is BaseTest {
         manager = new UniV4Inspector();
         router = new RouterActor(manager);
         vm.deal(address(router), 100 ether);
        vm.deal(address(manager), 10 ether);

         token = new MockERC20();
         token.mint(address(router), 1_000_000_000e18);
@@ -1048,4 +1049,37 @@ contract AngstromL2Test is BaseTest {
         vm.expectRevert(AngstromL2.IncompatiblePoolConfiguration.selector);
         angstrom.initializeNewPool(key, INIT_SQRT_PRICE, 0, 0);
     }

    function test_swapWithFees() public {
        PoolKey memory key = initializePoolWithFee(address(token), 1, 1,
            1000, 1000, 1000);
        addLiquidity(key, -20, 20, 10e18);
```

```
        setPriorityFee(0.5 gwei);
        router.swap(key, false, -50_000e18, int24(6).getSqrtPriceAtTick());
    }

    function initializePoolWithFee(
        address asset1,
        int24 tickSpacing,
        int24 startTick,
        uint24 swapFee,
        uint24 creatorSwapFeeE6,
        uint24 creatorTaxFeeE6
    )
        internal
        returns (PoolKey memory key)
    {
        require(asset1 ≠ address(0), "Token cannot be address(0)");

        key = PoolKey({
            currency0: Currency.wrap(address(0)),
            currency1: Currency.wrap(asset1),
            fee: swapFee,
            tickSpacing: tickSpacing,
            hooks: IHooks(address(angstrom))
        });

        vm.prank(hookOwner);
        angstrom.initializeNewPool(key, TickMath.getSqrtPriceAtTick(
            startTick), creatorSwapFeeE6, creatorTaxFeeE6);
    }
}
```

results in

```
[Revert] CurrencyNotSettled()
    □   □□ ← [Revert] CurrencyNotSettled()
    □□ ← [Revert] CurrencyNotSettled()
```

Turning on `afterSwapReturnDelta` fixes the test.

Recommendations:

```
permissions.afterSwapReturnDelta = true;
```

Sorella: Resolved with @d79a87b...

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] `beforeInitialization()` checks are bypassed due to `noSelfCall` modifier

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

AngstromL2.sol#L179, AngstromL2.sol#L186-L196

Description:

`beforeInitialize()` isn't called due to the `noSelfCall` modifier, that prevents calling a hook if they initiated the action.

```
modifier noSelfCall(IHooks self) {
  if (msg.sender ≠ address(self)) {
      _;
  }
}
```

As such, the incompatible pool config checks are bypassed.

POC

```
function test_incorrectConfigCheck() public {
    PoolKey memory key = PoolKey({
        currency0: Currency.wrap(address(1)),
        currency1: Currency.wrap(address(token)),
        fee: POOLS_MUST_HAVE_DYNAMIC_FEE ? LPFeeLibrary.DYNAMIC_FEE_FLAG :
    0,
        tickSpacing: 10,
        hooks: IHooks(address(angstrom))
```

```
    });

    vm.prank(hookOwner);
    angstrom.initializeNewPool(key, TickMath.getSqrtPriceAtTick(3), 0, 0);
}
```

Recommendations:

The `beforeInitialize()` hook permission is redundant, can be refactored into an internal function.

```
diff --git a/src/AngstromL2.sol b/src/AngstromL2.sol
index 8486b5d..e2febf7 100644
-- a/src/AngstromL2.sol
++ b/src/AngstromL2.sol
@@ -11,8 +11,7 @@ import {
     IBeforeSwapHook,
     IAfterSwapHook,
     IAfterAddLiquidityHook,
    IAfterRemoveLiquidityHook,
    IBeforeInitializeHook
    IAfterRemoveLiquidityHook
 } from "./interfaces/IHooks.sol";
 import {IFlashBlockNumber} from "./interfaces/IFlashBlockNumber.sol";
 import {IFactory} from "./interfaces/IFactory.sol";
@@ -39,7 +38,6 @@ import {tuint256, tbytes32} from
     "transient-goodies/TransientPrimitives.sol";
 contract AngstromL2 is
     UniConsumer,
     Ownable,
    IBeforeInitializeHook,
     IBeforeSwapHook,
     IAfterSwapHook,
     IAfterAddLiquidityHook,
@@ -176,6 +174,7 @@ contract AngstromL2 is
         if (!(creatorSwapFeeE6 ⩽ MAX_CREATOR_SWAP_FEE_E6))
             revert CreatorFeeExceedsMaximum();
         if (!(creatorTaxFeeE6 ⩽ MAX_CREATOR_TAX_FEE_E6))
             revert CreatorFeeExceedsMaximum();
         feeConfiguration.isInitialized = true;
        _checkInitialization(key);
         UNI_V4.initialize(key, sqrtPriceX96);
         feeConfiguration.creatorSwapFeeE6 =
```

```
                creatorSwapFeeE6.toUint24();
            feeConfiguration.creatorTaxFeeE6 =
                creatorTaxFeeE6.toUint24();
@@ -183,16 +182,9 @@ contract AngstromL2 is
             .recordPoolCreationAndGetStartingProtocolFee(key,
    creatorSwapFeeE6, creatorTaxFeeE6);
     }


    function beforeInitialize(address sender, PoolKey calldata key, uint160)
        external
        view
        returns (bytes4)
    {
        _onlyUniV4();
        if (sender != address(this)) revert Unauthorized();
    function _checkInitialization(PoolKey calldata key) internal pure {
            if (key.currency0.toId() != NATIVE_CURRENCY_ID)
                revert IncompatiblePoolConfiguration();
            if (LPFeeLibrary.isDynamicFee(key.fee))
                revert IncompatiblePoolConfiguration();
        return this.beforeInitialize.selector;
    }

     function afterAddLiquidity(
diff --git a/src/hook-config.sol b/src/hook-config.sol
index 1f5313a..3e076f1 100644
-- a/src/hook-config.sol
++ b/src/hook-config.sol
@@ -6,8 +6,6 @@ import {Hooks} from "v4-core/src/libraries/Hooks.sol";
 bool constant POOLS_MUST_HAVE_DYNAMIC_FEE = false;

 function getRequiredHookPermissions()
    pure returns (Hooks.Permissions memory permissions) {

    permissions.beforeInitialize = true; // To constrain that this is an ETH
        pool

     permissions.afterAddLiquidity = true; // To tax liquidity additions
    that may be JIT
     permissions.afterAddLiquidityReturnDelta = true; // To charge the JIT
    liquidity MEV tax.
```

Sorella: Resolved with @359ff6e...

Zenith: Verified. `beforeInitialize()` now reverts, to prevent others from permissionlessly using this hook.

[M-2] Liquidity ranges deltas are not rounded in the right direction

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- AngstromL2
- CompensationPriceFinder

Description:

The liquidity deltas calculations in the following functions always round down:

- `_zeroForOneCreditRewards()`
- `_oneForZeroCreditRewards()`
- `getZeroForOne()`
- `getOneForZero()`

The protocol rewards more taxes to LPs that were closer to the price before the swap, according to this:

- `delta1` should be rounded up in `_zeroForOneCreditRewards()` and `getZeroForOne()`
- `delta0` should be rounded up in `_oneForZeroCreditRewards()` and `getOneForZero()`

POC

This test results in a revert for underflow here because `delta1.divX96(pstarX96) < delta0` due to rounding down.

Before running the test make sure this issue is fixed.

```
function test_delta1WrongRoundingDirection() public {
    token.mint(address(router), 1e50);
    vm.deal(address(router), 1e50);

    int24 startTick = TickMath.MIN_TICK + 400000;
    PoolKey memory key = initializePool(address(token), 1, startTick);
```

```
    for(int24 i = 0; i < 80; i++) {
        addLiquidity(key, startTick-10-i*10, startTick-i*10, 100e18);
    }

    setPriorityFee(1 gwei);

    router.swap(key, true, 1e14, TickMath.MIN_SQRT_PRICE + 1);
}
```

Recommendations:

Rounding up increases `rangeReward` which is then subtracted from `lpCompensationAmount` here and here. This risks underflowing the `lpCompensationAmount`, maybe there's a better way to fix the revert shown in the test.

Otherwise:

- `delta1` should be rounded up in `_zeroForOneCreditRewards()` and `getZeroForOne()`
- `delta0` should be rounded up in `_oneForZeroCreditRewards()` and `getOneForZero()`

Sorella: Decided to take a different approach than rounding up the division, make sure that pstarX96 is at least 1 for both directions and also use saturating subtraction to ensure that the value never underflows: Resolved with @6450c9b...

Zenith: Verified.

4.3 Low Risk

A total of 4 low risk findings were identified.

[L-1] Fee mutators may cause total swap / tax fees to exceed 100%

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

AngstromL2.sol#L125-L137

Description:

There are initialization checks to ensure that the total creator and protocol swap / tax fees don't exceed 100%, but the fee mutators lack these checks. There actually exists protocol caps `MAX_PROTOCOL_SWAP_FEE_E6` and `MAX_PROTOCOL_TAX_FEE_E6`, but the former is only applied on initialization, while the latter is unused entirely.

It's more crucial for the total tax fee to be **strictly less than 100%**, because it leads to 0 `lpCompensationAmount` otherwise, which means `_blockOfLastTopOfBlock` isn't set to the (flash) block number.

```
if (!isTopOfBlock || lpCompensationAmount == 0) {
  return (this.afterSwap.selector, hookDeltaUnspecified);
}
_blockOfLastTopOfBlock = blockNumber; // ⟵ _blockOfLastTopOfBlock not set
    for 0 lpCompensationAmount
```

and thus JIT and swap tax will continue to be charged for subsequent LP or swap actions in the same block.

Recommendations:

Either ensure that the total creator and protocol swap / tax fees do not exceed 100%, or apply `MAX_PROTOCOL_SWAP_FEE_E6` / `MAX_PROTOCOL_TAX_FEE_E6` caps on the setters. Note that the existing value of `MAX_PROTOCOL_TAX_FEE_E6` at 75% may have to be adjusted down,

because `MAX_CREATOR_TAX_FEE_E6` is at 50%, bringing their theoretical total to 125%.

Sorella: Resolved with @2c07550...

Zenith: Verified.

[L-2] Rewards are lost on single-tick swaps with no active liquidity

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- AngstromL2

Description:

The protocol allows swapping exactly `0` ETH by passing as input an amount of `ETH` equal to the swap tax.

There is an edge case where rewards are lost when swapping `0` ETH on pools whose current tick has no active liquidity, this happens because the swap starts and ends in the same tick and because there is no liquidity the liquidityBeforeSwap state variable will return `0`:

```
function test_ZeroSwapOnNonLiquidityTick() public {
    //pool initial tick is set to 350
    PoolKey memory key = initializePool(address(token), 2, 350);

    //Add liqudiity in range 0,300
    router.modifyLiquidity(key, 0, 300, int256(uint256(100e18)),
    bytes32(0));

    //Swap zero ETH
    setPriorityFee(100 gwei);
    router.swap(key, true, -4.9e17, TickMath.MIN_SQRT_PRICE + 1);

    //Only position in the protocol doesn't have rewards
    uint256 rewards = angstrom.getPendingPositionRewards(key,
    address(router), 0, 300, bytes32(0));
    assertEq(rewards, 0);
}
```

Recommendations:

Because no liquidity contributed to the swap the tax can be given to the pool creator or the protocol.

Sorella: Acknowledged, deemed not worth the complexity to capture this loss. User slippage checks should always prevent this.

[L-3] `pstarX96` rounds down to zero on low ticks ranges leading to reverts for division by zero

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- AngstromL2.sol

Description:

The calculation of `pstarX96` can round down to `0` in:

- _zeroForOneCreditRewards()
- _oneForZeroCreditRewards

If `pstarX96` is rounded down to `0` both functions will revert because of division by `0`:

- here for `_zeroForOneCreditRewards()` during `rangeReward` calculations
- here for `_oneForZeroCreditRewards()`during `rangeReward` calculations

The variable `pstarX96` gets rounded down to `0` when `pstarSqrtX96^2 < 2^96`, `pstarSqrtX96 < 281474976710656`, so this only happens on ticks below `-665455`.

To note that squaring tick boundaries prices also leads to rounding to `0`:

- In `getZeroForOne()`: here and here
- In `getOneForZero()`: here and here

POC

This test shows that `pastrX96` being rounded to zero leads to a revert in _zeroForOneCreditRewards() calculation because of division by `0`.

Before running it make sure this issue is fixed:

```
function test_pstarX96ZeroRevert() public {
    token.mint(address(router), 1e50);
    vm.deal(address(router), 1e50);
    int24 startTick = TickMath.MIN_TICK + 200000;
```

```
    PoolKey memory key = initializePool(address(token), 1, startTick);

    addLiquidity(key, startTick-2000, startTick+100, 1000e18);

    setPriorityFee(1 gwei);
    swap(key, true, 1e13, allPositions);
}
```

Recommendations:

These scenarios regard edge cases on really low ticks ranges and because `token0` is always `ETH` this shouldn't pose a problem in realistic circumstances.

Otherwise TBD, likely either:

- Implement a liquidity distribution strategy for the edge cases of `pstarX96` being zero
- Increase precision

Sorella: Resolved with @6450c9b... where a floor of `1` is applied.

Zenith: Verified.

[L-4] JIT tax can be avoided by first doing a small swap

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- AngstromL2.sol#L198-L242

Description:

While the swap tax is unavoidable (fixed price that scales with priority fee) and is incurred on the first swap performed, subsequent actions (LP creations modifications & swaps) will skip charging the swap and JIT tax when `_blockOfLastTopOfBlock` is updated to `blockNumber`.

Hence, consider a JIT liquidity sandwich: add liquidity -> swap -> remove liquidity. The swapper incurs both the swap and JIT tax fees with this bundle, but by swapping a negligible amount first, he only incurs the swap tax.

Also, the JIT tax is 4x the swap tax, so one is economically incentivised to do a swap prior to liquidity modifications.

Recommendations:

To discuss if current design is acceptable. Maybe charge heavily discounted fees instead of 0 for subsequent actions.

Sorella: Resolved by ensuring the tax is always charged, regardless of position in block: @b6bd4e2... this way you can't game it by just doing some swaps at the beginning.JIT becomes a similar problem so we decide to always enable it as well but bring it down to being 1.5x the swap tax rather than 4x to reduce the cost for average LPs but also ensure being at the top of the block is disincentivized

Zenith: Verified, both the swap and JIT tax will always be charged now (except for claiming rewards only).

4.4 Informational

A total of 6 informational findings were identified.

[I-1] Native ID conversion to currency can be replaced with `NATIVE_CURRENCY` constant

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- AngstromL2.sol#L213, AngstromL2.sol#L233

Description:

`CurrencyLibrary.fromId(NATIVE_CURRENCY_ID)`, can be replaced with the already declared `NATIVE_CURRENCY`.

Recommendations:

```
UNI_V4.take(CurrencyLibrary.fromId
        (NATIVE_CURRENCY_ID), FACTORY, taxAmountInEther);
UNI_V4.take(NATIVE_CURRENCY, FACTORY, taxAmountInEther);
```

Sorella: Resolved with @4702d84... and @6db1a73...

Zenith: Verified.

[I-2] Redundant checks on `params.liquidityDelta`

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- PoolRewards.sol#L81, PoolRewards.sol#L117

Description:

The referenced lines are redundant, because it's checked by the `PoolManager` to decide whether `afterAddLiquidity` or `afterRemoveLiquidity` should be called.

Note that 0 `liquidityDelta` (eg. poking positions to collect rewards) falls under the else branch, calling the `afterRemoveLiquidity()` hook.

Recommendations:

Remove the referenced lines.

Sorella: Acknowledged. Keep the referenced lines as sanity check.

[I-3] `_oneForZeroCreditRewards()` calculates rewards of ranges with 0 liquidity

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- AngstromL2.sol

Description:

The function _oneForZeroCreditRewards() calculates the rewards of ranges with `0` liquidity:

```
if (tickNext <= lastTick || liquidity == 0) {
// ... snip ...
}
```

Recommendations:

In _oneForZeroCreditRewards():

```
if (tickNext <= lastTick && liquidity != 0) {
// ... snip ...
}
```

Sorella: Resolved with @d53cc19...

Zenith: Verified.

[I-4] The `updateAfterLiquidityAdd()` function doesn't initialize the reward accumulator of uninitialized ticks

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- PoolRewards.sol

Description:

The function updateAfterLiquidityAdd attempts to initialize the `rewardGrowthOutsideX128` value of unintialized ticks to `self.globalGrowthX128`.

The `!pm.isInitialized` if conditions are never true because ticks are already initialized during the call flow of the `afterAddLiquidity` hook.

Recommendations:

Not initializing the growth accumulator seems to have no impact.

Sorella: Addressed in @cd0ac3c....

Zenith: Verified. The initialization values themselves bear no meaning, so the logic has been simplified to set `feeGrowthInside = getGrowthInsideX128(currentTick, params.tickLower, params.tickUpper);`

[I-5] Indirectly claiming rewards is subject to MEV tax

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- AngstromL2.sol#L229-L234

Description:

An indirect method of only claiming rewards is by poking positions (zero liquidity modification). However, this method is subject to the MEV tax if it's the 1st action of a block.

POC:

```
function test_indirectlyClaimingRewardsSubjectToTax() public {
    PoolKey memory key = initializePool(address(token), 1, 1);
    addLiquidity(key, -20, 20, 100e18);

    uint256 priorityFee = 0.5 gwei;
    setPriorityFee(priorityFee);
    router.swap
        (key, false, -50_000e18, int24(6).getSqrtPriceAtTick());
    assertGt(getRewards(key, -20, 20), 0);

    priorityFee = 0.05 gwei;

    uint256 snapshotId = vm.snapshotState();
    BalanceDelta removeDeltaNoTax;
    BalanceDelta removeDeltaWithTax;
    {
        // call remove liquidity with 0 liquidity delta to get rewards
        (removeDeltaNoTax, ) =
            router.modifyLiquidity(key, -20, 20, 0, "");
        // rewards collected strictly in token0 (ETH)
        assertGt(removeDeltaNoTax.amount0(), 0);
        assertEq(removeDeltaNoTax.amount1(), 0);
        vm.revertToState(snapshotId);
```

```
    }

    {
        vm.roll(block.number + 1);
        setPriorityFee(priorityFee);
        (removeDeltaWithTax, ) =
            router.modifyLiquidity(key, -20, 20, 0, "");
        // rewards collected strictly in token0 (ETH)
        assertGt(removeDeltaWithTax.amount0(), 0);
        assertEq(removeDeltaWithTax.amount1(), 0);
    }

    assertEq(
        removeDeltaNoTax.amount0(),
        removeDeltaWithTax.amount0() + int128
            (uint128(angstrom.getJitTaxAmount(priorityFee)))
    );
}
```

Recommendations:

Consider charging 0 fees for position pokes. Alternatively, have a method to directly claim rewards.

Sorella: Resolved with @731738d...

Zenith: Verified, claiming rewards are tax exempt.

[I-6] Redundancies

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- AngstromL2.sol#L57
- AngstromL2.sol#L59
- AngstromL2Factory.sol#L50
- AngstromL2Factory.sol#L52

Description:

`NegationOverflow()`, `AttemptingToWithdrawLPRewards()`, `MAX_DEFAULT_PROTOCOL_FEE_MULTIPLE_E6` and `MAX_PROTOCOL_TAX_FEE_E6` are defined but unused.

Recommendations:

Remove the referenced lines.

Sorella: Resolved with @4702d84...

Zenith: Verified.