Stake.Link Staked Link Rebase Batching Audit Report

Prepared by Cyfrin

Version 1.0

Lead Auditors

Kage

March 23, 2026

Contents

1 About Cyfrin

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

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the in-scope code being audited.

3 Risk Classification

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

4 Protocol Summary

5 Audit Scope

The audit scope was limited to PR 178 that impacts changes to the following contracts:

```
contracts/linkStaking/CommunityVCS.sol
contracts/linkStaking/base/VaultControllerStrategy.sol
```

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the Staked Link Rebase Batching code provided by Stake.Link. The findings consist of 1 Medium & 1 Low severity issues.

Summary

Project Name	Staked Link Rebase Batching
Repository	contracts
Commit	b93e4c9499b3...
Audit Timeline	March 23rd - March 24th, 2026
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	0
Gas Optimizations	0
Total Issues	2

Summary of Findings

[M-1] `CommunityVCS::updateDeposits` reverts during batch updates, blocking `RebaseController` from updating rewards for all strategies	Open
[L-1] `CommunityVCS::canDeposit` and `CommunityVCS::canWithdraw` are inconsistent with the newly introduced batch update logic	Open

7 Findings

7.1 Medium Risk

7.1.1 `CommunityVCS::updateDeposits` reverts during batch updates, blocking `RebaseController` from updating rewards for all strategies

Description: `CommunityVCS::updateDeposits` now reverts with `DepositUpdateNotReady` when batching is enabled and not all vault batches have been processed:

```
// CommunityVCS.sol
function updateDeposits(
    bytes calldata _data
)
    external
    override
    onlyStakingPool
    returns (int256 depositChange, address[] memory receivers, uint256[] memory amounts)
{
    if (vaultsPerBatch == 0) {
        depositChange = getDepositChange();
    } else {
        // @audit - reverts if batches are incomplete
        if (currentVaultIndex < vaults.length) revert DepositUpdateNotReady();
        ...
    }
    ...
}
```

`RebaseController::_updateRewards` always passes ALL strategy indexes when triggering a reward update:

```
// RebaseController.sol
function _updateRewards(bytes memory _data) private {
    address[] memory strategies = stakingPool.getStrategies();
    uint256[] memory strategyIdxs = new uint256[](strategies.length);

    for (uint256 i = 0; i < strategies.length; ++i) {
        strategyIdxs[i] = i; // @audit - includes ALL strategies, including CommunityVCS
    }

    stakingPool.updateStrategyRewards(strategyIdxs, _data); // @audit - single call, any revert fails
    ↪  all
}
```

Inside `StakingPool::_updateStrategyRewards`, each strategy's `updateDeposits` is called in a loop with no try/catch:

```
// StakingPool.sol
function _updateStrategyRewards(uint256[] memory _strategyIdxs, bytes memory _data) private {
    ...
    for (uint256 i = 0; i < _strategyIdxs.length; ++i) {
        IStrategy strategy = IStrategy(strategies[_strategyIdxs[i]]);
        // @audit - if CommunityVCS reverts here, the entire loop reverts
        (int256 depositChange, ...) = strategy.updateDeposits(_data);
        ...
    }
    ...
}
```

When CommunityVCS has batching enabled and its batch is not yet complete, any call to `RebaseController::updateRewards` or `RebaseController::reopenPool` reverts, because the `updateDeposits` call on CommunityVCS reverts and propagates up through the entire call chain.

Impact: During any CommunityVCS batch update window, **stLINK rebasing is completely blocked for all users** — not just CommunityVCS stakers, but also OperatorVCS stakers and any other strategy's stakers. This is because `RebaseController` updates all strategies atomically.

The batch window spans multiple transactions (one per batch). If the protocol has many vaults and small batch sizes, this window could span many blocks. During this entire period:

- No reward distribution occurs for any strategy
- `reopenPool` cannot execute (it calls `_updateRewards` internally), meaning the pool cannot be reopened after an emergency pause if a batch is in progress

Proof of Concept: Run

```
yarn hardhat test test/audit/poc-H01-rebase-blocked.test.ts --network hardhat
```

```
import { ethers } from 'hardhat'
import { assert, expect } from 'chai'
import {
  toEther,
  deploy,
  deployUpgradeable,
  getAccounts,
  setupToken,
  fromEther,
  deployImplementation,
} from '../utils/helpers'
import {
  ERC677,
  StrategyMock,
  StakingPool,
  RebaseController,
  PriorityPool,
  SecurityPool,
  CommunityVCS,
  StakingMock,
  StakingRewardsMock,
} from '../../typechain-types'
import { loadFixture } from '@nomicfoundation/hardhat-network-helpers'

const unbondingPeriod = 28 * 86400
const claimPeriod = 7 * 86400

const encodeVaults = (vaults: number[]) => {
  return ethers.AbiCoder.defaultAbiCoder().encode(['uint64[]'], [vaults])
}

describe('PoC H-01: RebaseController blocked during CommunityVCS batch', () => {
  async function deployFixture() {
    const { signers, accounts } = await getAccounts()
    const adrs: any = {}

    // Deploy LINK token
    const token = (await deploy('contracts/core/tokens/base/ERC677.sol:ERC677', [
      'Chainlink',
      'LINK',
      1000000000,
    ])) as ERC677
    adrs.token = await token.getAddress()
    await setupToken(token, accounts)

    // Deploy StakingPool
    const stakingPool = (await deployUpgradeable('StakingPool', [
      adrs.token,
```

```
    'LinkPool LINK',
    'lpLINK',
    [[accounts[4], 1000]],
    toEther(10000),
  ])) as StakingPool
  adrs.stakingPool = await stakingPool.getAddress()

  // Deploy PriorityPool
  const priorityPool = (await deployUpgradeable('PriorityPool', [
    adrs.token,
    adrs.stakingPool,
    accounts[0],
    toEther(100),
    toEther(1000),
    false,
  ])) as PriorityPool
  adrs.priorityPool = await priorityPool.getAddress()

  // Deploy SecurityPool
  const securityPool = (await deployUpgradeable('SecurityPool', [
    adrs.token,
    'name',
    'symbol',
    accounts[0],
    3000,
    10,
    100,
  ])) as SecurityPool
  adrs.securityPool = await securityPool.getAddress()

  // Deploy RebaseController
  const rebaseController = (await deploy('RebaseController', [
    adrs.stakingPool,
    adrs.priorityPool,
    adrs.securityPool,
    accounts[0], // emergencyPauser
    accounts[0], // rewardsUpdater
  ])) as RebaseController
  adrs.rebaseController = await rebaseController.getAddress()

  // Deploy StrategyMock (simulates OperatorVCS - a healthy strategy that should not be blocked)
  const strategyMock = (await deployUpgradeable('StrategyMock', [
    adrs.token,
    adrs.stakingPool,
    toEther(5000),
    toEther(0),
  ])) as StrategyMock
  adrs.strategyMock = await strategyMock.getAddress()

  // Deploy CommunityVCS (the strategy with batching)
  const rewardsController = (await deploy('StakingRewardsMock', [
    adrs.token,
  ])) as StakingRewardsMock
  adrs.rewardsController = await rewardsController.getAddress()

  const stakingController = (await deploy('StakingMock', [
    adrs.token,
    adrs.rewardsController,
    toEther(10),
    toEther(100),
    toEther(10000),
    unbondingPeriod,
    claimPeriod,
```

```
  ])) as StakingMock
  adrs.stakingController = await stakingController.getAddress()

  let vaultImplementation = await deployImplementation('CommunityVault')
  const vaultDepositController = await deploy('VaultDepositController')

  const communityVCS = (await deployUpgradeable(
    'CommunityVCS',
    [
      adrs.token,
      adrs.stakingPool, // stakingPool is the actual StakingPool (not accounts[0])
      adrs.stakingController,
      vaultImplementation,
      [[accounts[4], 500]],
      9000,
      toEther(100),
      10,
      20,
      vaultDepositController.target,
    ],
    { unsafeAllow: ['delegatecall'] }
  )) as CommunityVCS
  adrs.communityVCS = await communityVCS.getAddress()

  // Register strategies with StakingPool - StrategyMock at index 0, CommunityVCS at index 1
  await stakingPool.addStrategy(adrs.strategyMock)
  await stakingPool.addStrategy(adrs.communityVCS)

  await stakingPool.setPriorityPool(accounts[0])
  await stakingPool.setRebaseController(adrs.rebaseController)
  await priorityPool.setRebaseController(adrs.rebaseController)
  await securityPool.setRebaseController(adrs.rebaseController)

  // Fund rewards controller
  await token.transfer(adrs.rewardsController, toEther(10000))

  // Configure CommunityVCS batching
  await communityVCS.setDepositUpdater(accounts[0])

  // Deposit into the pool - funds go to strategyMock first (index 0)
  await token.approve(adrs.stakingPool, ethers.MaxUint256)
  await stakingPool.deposit(accounts[0], toEther(1000), ['0x', encodeVaults([])])

  return {
    signers,
    accounts,
    adrs,
    token,
    stakingPool,
    priorityPool,
    securityPool,
    rebaseController,
    strategyMock,
    communityVCS,
    rewardsController,
  }
}

it('POC: updateRewards reverts when CommunityVCS batch is in progress', async () => {
  const { accounts, adrs, token, rebaseController, strategyMock, communityVCS } =
    await loadFixture(deployFixture)

  // Verify both strategies are registered
```

```
  const strategies = await (await ethers.getContractAt('StakingPool',
  ↪  adrs.stakingPool)).getStrategies()
  assert.equal(strategies.length, 2, 'should have 2 strategies')
  assert.equal(strategies[0], adrs.strategyMock, 'strategy 0 should be StrategyMock')
  assert.equal(strategies[1], adrs.communityVCS, 'strategy 1 should be CommunityVCS')

  // Simulate rewards accruing on the mock strategy (representing OperatorVCS)
  await token.transfer(adrs.strategyMock, toEther(50))
  const mockRewardsBefore = fromEther(await strategyMock.getDepositChange())
  assert.equal(mockRewardsBefore, 50, 'StrategyMock should have 50 LINK in rewards')

  // 1: updateRewards works BEFORE batching is enabled
  await rebaseController.updateRewards('0x')
  assert.equal(fromEther(await strategyMock.getDepositChange()), 0, 'StrategyMock rewards should be
  ↪  processed')

  // 2: Enable batching on CommunityVCS
  await communityVCS.setVaultsPerBatch(5)
  assert.equal(Number(await communityVCS.vaultsPerBatch()), 5)

  // More rewards accrue on mock strategy
  await token.transfer(adrs.strategyMock, toEther(30))
  assert.equal(fromEther(await strategyMock.getDepositChange()), 30, 'StrategyMock has 30 LINK pending
  ↪  rewards')

  // 3: Start a batch on CommunityVCS (process only first 5 of 20 vaults)
  await communityVCS.updateVaultDeposits()
  assert.equal(Number(await communityVCS.currentVaultIndex()), 5, 'batch started, 5 vaults processed')

  // 4: Try to update rewards - THIS SHOULD REVERT
  // RebaseController passes all strategy indexes, CommunityVCS reverts with DepositUpdateNotReady
  await expect(rebaseController.updateRewards('0x')).to.be.revertedWithCustomError(
    communityVCS,
    'DepositUpdateNotReady'
  )

  // 5: Verify StrategyMock rewards are STILL pending (blocked by CommunityVCS revert)
  assert.equal(
    fromEther(await strategyMock.getDepositChange()),
    30,
    'StrategyMock rewards are BLOCKED - 30 LINK still pending despite being a healthy strategy'
  )

  // 6: Complete all batches on CommunityVCS
  await communityVCS.updateVaultDeposits() // vaults 5-9
  await communityVCS.updateVaultDeposits() // vaults 10-14
  await communityVCS.updateVaultDeposits() // vaults 15-19
  assert.equal(Number(await communityVCS.currentVaultIndex()), 20, 'all batches complete')

  // 7: NOW updateRewards works again
  await rebaseController.updateRewards('0x')
  assert.equal(
    fromEther(await strategyMock.getDepositChange()),
    0,
    'StrategyMock rewards finally processed after CommunityVCS batch completed'
  )
})

it('POC: reopenPool also blocked during CommunityVCS batch', async () => {
  const { accounts, adrs, token, rebaseController, strategyMock, communityVCS, priorityPool } =
    await loadFixture(deployFixture)

  // Simulate a loss to trigger pool pause (before batching is enabled)
```

```
    await token.transfer(adrs.strategyMock, toEther(100))
    await rebaseController.updateRewards('0x')
    await strategyMock.simulateSlash(toEther(50))
    assert.isBelow(fromEther(await strategyMock.getDepositChange()), 0, 'StrategyMock shows a loss')

    // Pause the pool
    await rebaseController.pausePool()
    assert.equal(Number(await priorityPool.poolStatus()), 2, 'pool should be closed')

    // NOW enable batching and start a batch on CommunityVCS
    await communityVCS.setVaultsPerBatch(5)
    await communityVCS.updateVaultDeposits()
    assert.equal(Number(await communityVCS.currentVaultIndex()), 5, 'batch in progress')

    // Try to reopen the pool - reverts because reopenPool calls _updateRewards internally
    await expect(rebaseController.reopenPool('0x')).to.be.revertedWithCustomError(
      communityVCS,
      'DepositUpdateNotReady'
    )

    // Pool remains closed - cannot be reopened until CommunityVCS batch completes
    assert.equal(
      Number(await priorityPool.poolStatus()),
      2,
      'pool is STILL closed - reopenPool blocked by CommunityVCS batch'
    )
  })
})
```

Recommended Mitigation: Consider having `CommunityVCS::updateDeposits` return zero change instead of reverting when batches are incomplete:

```
function updateDeposits(
        bytes calldata _data
    )
        external
        override
        onlyStakingPool
        returns (int256 depositChange, address[] memory receivers, uint256[] memory amounts)
{
        if (vaultsPerBatch == 0) {
            // batching not configured, use default behavior
            depositChange = getDepositChange();
        } else {
-               if (currentVaultIndex < vaults.length) revert DepositUpdateNotReady();
+               if (currentVaultIndex < vaults.length) {
+                   // Batch not complete, report no change
+                   return (0, new address[](0), new uint256[](0));
                }
                ...
       }
}
```

Stake.Link: Cyfrin:

7.2 Low Risk

7.2.1 `CommunityVCS::canDeposit` and `CommunityVCS::canWithdraw` are inconsistent with the newly introduced batch update logic

Description: `CommunityVCS::canDeposit` and `CommunityVCS::canWithdraw` inherit from `Strategy.sol` without overriding them. These view functions check accounting state but have no awareness of the `notDuringDepositUpdate` batch lock:

```
//Strategy.sol
function canDeposit() public view virtual returns (uint256) {
    uint256 deposits = getTotalDeposits();
    if (deposits >= getMaxDeposits()) {
        return 0;
    } else {
        return getMaxDeposits() - deposits; // @audit returns > 0 even during batch lock
    }
}

function canWithdraw() public view virtual returns (uint256) {
    uint256 deposits = getTotalDeposits();
    if (deposits <= getMinDeposits()) {
        return 0;
    } else {
        return deposits - getMinDeposits(); // @audit returns > 0 even during batch lock
    }
}
```

`StakingPool::_depositLiquidity` relies on `canDeposit()` to decide whether to call `deposit()`:

```
//StakingPool.sol
function _depositLiquidity(bytes[] calldata _data) private {
    ...
    for (uint256 i = 0; i < strategies.length; i++) {
        IStrategy strategy = IStrategy(strategies[i]);
        uint256 strategyCanDeposit = strategy.canDeposit(); // @audit returns > 0 for locked
        ↪  CommunityVCS

        if (strategyCanDeposit >= toDeposit) {
            strategy.deposit(toDeposit, strategyData); // @audit reverts with DepositUpdateInProgress
            break;
        } else if (strategyCanDeposit > 0) {
            strategy.deposit(strategyCanDeposit, strategyData); // @audit reverts
            ...
        }
    }
}
```

When `canDeposit` returns `> 0` but deposit reverts due to the batch lock, the entire `StakingPool.deposit()` transaction reverts. The same issue applies to `_withdrawLiquidity` using `canWithdraw`.

Impact: During any CommunityVCS batch window, if CommunityVCS has deposit room or withdrawable balance, `StakingPool::deposit()` and `StakingPool::withdraw()` revert when they reach `CommunityVCS` in the strategy loop.

The blocking is temporary (batch window duration), no funds are lost, and users can retry after the batch completes.

Proof of Concept: Add the following test:

```
  it('POC L-01: canDeposit reports capacity during batch, causing StakingPool deposit revert', async ()
  ↪  => {
    const { accounts, adrs, token, stakingPool, communityVCS, strategyMock } =
      await loadFixture(deployFixture)
```

```
    // CommunityVCS reports deposit room available
    assert.isAbove(Number(await communityVCS.canDeposit()), 0, 'CommunityVCS reports deposit room')

    // Enable batching and start a batch
    await communityVCS.setVaultsPerBatch(5)
    await communityVCS.updateVaultDeposits()
    assert.equal(Number(await communityVCS.currentVaultIndex()), 5, 'batch in progress')

    // BUG: canDeposit STILL reports room - unaware of batch lock
    assert.isAbove(
      Number(await communityVCS.canDeposit()),
      0,
      'canDeposit() still reports room during batch lock'
    )

    // Deposit enough to fill strategyMock (index 0, max 5000) so overflow reaches CommunityVCS
    // StakingPool already has ~1000 deposited from fixture, strategyMock has ~4000 room
    const depositAmount = toEther(4500)
    await token.approve(adrs.stakingPool, depositAmount)

    // StakingPool.deposit → _depositLiquidity → fills strategyMock → tries CommunityVCS
    // canDeposit() returns > 0 → calls deposit() → REVERTS with DepositUpdateInProgress
    await expect(
      stakingPool.deposit(accounts[0], depositAmount, ['0x', encodeVaults([])])
    ).to.be.revertedWithCustomError(communityVCS, 'DepositUpdateInProgress')

    // The 4500 LINK deposit failed entirely - even the portion that could have gone to strategyMock
    // was rolled back because the revert propagates from CommunityVCS through the whole tx
    assert.equal(
      fromEther(await strategyMock.getTotalDeposits()),
      1000,
      'strategyMock deposits unchanged - entire tx reverted'
    )
  })
```

Recommended Mitigation: Consider override `canDeposit()` and `canWithdraw()` in `CommunityVCS` to return 0 during a batch update.

Stake.Link: Cyfrin: