Illuvium Staking V3 Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Immeas

ChainDefenders (0x539 & PeterSR)

October 21, 2025

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 solidity implementation of the contracts.

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

Illuvium Staking V3 provides an revenue distribution and staking system for ILV on Base. An operator converts inbound ETH/WETH to ILV via Aerodrome and distributes ILV across configured pools. Distribution happens in two phases per call:

1) fixed-BPS pools receive their share
2) remainder is allocated to reserve-weighted pools based on "units" that reflect either an ILV vault's `StakingVault::totalStaked` or an LP vault's proportional share of ILV reserves in the paired pool

Reserve weighting falls back to equal split when units are zero; rounding dust goes to the first eligible pool. Pools can be vaults (ILV is transferred and `StakingVault::notifyRewardAmount` is called) or directly claimable (ILV accrues to `L2RevenueDistributorV3::pendingIlv` and can be pulled by allowlisted claimers). Staking is handled by time-locked vaults with a per share accumulator that tracks and pays ILV rewards pro-rata to stake.

4.1 Actors and Roles

- **1. Actors:**
  - **Protocol team:** Operates distribution and configuration; upgrades contracts
  - **Vault stakers:** Stake ILV or LP tokens into vaults and earn ILV rewards
  - **Claim recipients:** Allowlisted callers that pull ILV accrued to claimable pools
- **2. Roles:**
  - **Distributor Admin (L2RevenueDistributorV3 – `ADMIN_ROLE`/`DEFAULT_ADMIN_ROLE`):** Adds/updates pools, sets BPS, toggles active flags, manages claim allowlists, pauses/unpauses, upgrades, and can withdraw ETH
  - **Distributor Operator (`OPERATOR_ROLE`):** Executes swaps and `L2RevenueDistributorV3::distribute`

- **Vault Admin (`ADMIN_ROLE`/`DEFAULT_ADMIN_ROLE`):** Pauses/unpauses, sets lock durations, manages who has `DISTRIBUTOR_ROLE`, and upgrades
- **Vault Distributor (`DISTRIBUTOR_ROLE`):** Calls `StakingVault::notifyRewardAmount` after ILV is transferred in

4.2 Key Components

- **L2RevenueDistributorV3:** Holds ILV and allocates it across pools. Supports:
  - **Swaps:** `swapExactETHForILV` and `swapExactWETHForILV` via Aerodrome router
  - **Pool types:**
    - **Vault:** `ilv.safeTransfer(pool.recipient, amount)` then `notifyRewardAmount(amount)`
    - **Claimable:** Increases `pendingIlv`; allowlisted claimers pull via `claimPool`
  - **Allocation logic:** Fixed-BPS first; reserve-weighted on remainder using:
    - **ILV vault units:** `totalStaked()` of the vault
    - **LP vault units:** `(ILV_reserve_in_pair * vaultLP / pair_totalSupply)`
  - **Guards:** BPS cap (10,000), pending claimable cannot be migrated to vault, pausable, reentrancy-guarded, UUPS upgradeable
- **StakingVault:** Time-locked staking for an ERC-20 (ILV or LP) with ILV rewards:
  - Users create per-stake locks; withdrawals require the lock to be expired
  - Rewards: Tracked via `accIlvPerShare`; `notifyRewardAmount` increments the accumulator pro-rata to `totalStaked`. We recommend the team prefer smaller, more frequent reward top-ups over infrequent large ones to reduce deposit-timing arbitrage. While not making them too small to avoid rounding loss
  - Users harvest with `claimRewards` (pulls accumulated ILV)
- **StakingVaultFactory:** Deploys new `StakingVault` ERC1967 proxies pointing to a shared implementation and initializes them with asset, ILV token, lock bounds, and admin

4.3 Centralization risk

Configuration, upgrades (UUPS), and operational controls are admin-gated. The distributor operator controls swaps and the timing/amount of distributions; vault admins designate who can post rewards. Compromise or misuse of these roles can impact reward flows, pool configuration, or upgrades. We recommend multisigs, timelocks, strong key management, and monitoring for role actions and distribution events.

5 Audit Scope

The audit scope was limited to:

```
src/L2RevenueDistributorV3.sol
src/StakingVault.sol
src/StakingVaultFactory.sol
```

6 Executive Summary

Over the course of 6 days, the Cyfrin team conducted an audit on the Illuvium Staking V3 smart contracts provided by Illuvium. In this period, a total of 16 issues were found.

During the audit no critical, high, or medium-severity issues were identified. 7 low-severity and several informational items were identified however. The most consequential lows are operational or allocation-logic flaws that could stall payouts or skew reward splits without creating irreversible loss.

The low severity findings included:

- Claimables can be over-committed by `L2RevenueDistributorV3::distribute`, temporarily DoS-ing `L2RevenueDistributorV3::claimPool` until top-up
- `ILV` can be stranded if sent to empty vaults before `StakingVault::notifyRewardAmount` accrues
- Reserve-weighted splits use spot reserves and are manipulable around `L2RevenueDistributorV3::distribute`
- Equal-split fallback may exclude vaults when units are zero
- Rounding/dust leaves tiny amounts unallocated

Informationals included: unused storage, missing events, code quality enhancements, input validations and sanity checks.

Summary

Project Name	Illuvium Staking V3
Repository	staking-contracts-v3
Commit	c78653ed5f2e...
Fix Commit	f69a4a277ff4...
Audit Timeline	Sep 2nd - Sep 9th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	7
Informational	8
Gas Optimizations	1
Total Issues	16

Summary of Findings

[L-1] Calling `StakingVault::notifyRewardAmount` on empty vault leaves ILV stuck	Resolved
[L-2] Deployment script requires unencrypted private key	Resolved
[L-3] Dust amounts of rewards get stuck in the Vault	Resolved
[L-4] Distributor can over-allocate claimables, causing temporary claim DoS	Resolved

[L-5] Reserve-weighted allocations use spot reserves	Acknowledged
[L-6] Dust rewards can become permanently locked	Resolved
[L-7] Equal distribution fallback logic error	Resolved
[I-1] Unused `factory` storage in `L2RevenueDistributorV3`	Resolved
[I-2] Lack of events emitted for important state changes	Resolved
[I-3] Redundant recomputation of accrued rewards in `StakingVault::_updateRewards`	Resolved
[I-4] Consider adding a minimum deposit amount in `StakingVault`	Resolved
[I-5] Missing sanity check for reasonable duration values	Resolved
[I-6] Missing `address(0)` check in `setDistributor`	Resolved
[I-7] Lack of validation of `Pool.underlyingToken == StakingVault.stakingAsset`	Resolved
[I-8] Inconsistent `recipient` handling for claimable pools	Resolved
[G-1] Unoptimised overflow check in `_assertFixedBpsCap`	Resolved

7 Findings

7.1 Low Risk

7.1.1 Calling `StakingVault::notifyRewardAmount` on empty vault leaves ILV stuck

Description: `StakingVault::notifyRewardAmount` is the vault's reward hook that updates the rewards-per-share accumulator (`accIlvPerShare`) so stakers can later claim ILV. It is called by `L2RevenueDistributorV3::_applyAllocation` after it has already transferred ILV to the vault:

```
if (pool.kind == PoolKind.Vault) {
    // Transfer ILV to the vault and notify
    ilv.safeTransfer(pool.recipient, amount);
    IStakingVaultMinimal(pool.recipient).notifyRewardAmount(amount);
} else {
```

If `notifyRewardAmount` is invoked while `totalStaked == 0`, `StakingVault::notifyRewardAmount` returns early and the transferred ILV is not added to `accIlvPerShare`:

```
function notifyRewardAmount(uint256 ilvAmount)
    external
    override
    nonReentrant
    whenNotPaused
    onlyRole(DISTRIBUTOR_ROLE)
{
    // If no rewards are provided, return
    if (ilvAmount == 0) return;
```

As a result, those tokens sit in the vault and are not claimable via normal flows; there is no built-in sweep/buffer to recover them later.

Impact: The funds are effectively stuck: neither users nor admins can redistribute or withdraw the stranded ILV using existing code paths. In practice, the only recovery is to upgrade the contract. Until then, the ILV remains idle in the vault.

Recommended Mitigation: * Vault-side buffer: Track `unallocatedRewards` and always add incoming amounts to it; when `totalStaked > 0`, fold the entire buffer into `accIlvPerShare`.

- Distributor-side skip/carry: Before transfering to a vault, check `totalStaked() > 0`. If zero, skip the transfer and record a per-pool carry to retry later (or divert to a claimable bucket).
- Hard fail on empty vault: In `notifyRewardAmount`, revert when `totalStaked == 0` to prevent accidental stranding (e.g., `require(totalStaked > 0, "No stakers")`). Together with per-pool try/catch or skip logic in the distributor to avoid whole-tx failures.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified; Vault-side buffer mitigation was implemented.

7.1.2 Deployment script requires unencrypted private key

Description: The Makefile's deployment targets require a raw private key to be supplied via an environment variable and pass it directly on the command line to 'forge':

```
# Guard (fails if PRIVATE_KEY not provided)
@if [ -z "$(PRIVATE_KEY)" ]; then \
    echo "$(RED)ERROR: PRIVATE_KEY not set$(NC)"; \
    exit 1; \
fi

# Usage (Sepolia)
forge script script/Deployer.s.sol:Deployer \
  --rpc-url $(BASE_SEPOLIA_RPC) \
  --private-key $(PRIVATE_KEY) \
  --broadcast \
  --verify \
  --etherscan-api-key $(BASESCAN_API_KEY) \
  -vvvv

# Usage (Mainnet)
forge script script/Deployer.s.sol:Deployer \
  --rpc-url $(BASE_RPC) \
  --private-key $(PRIVATE_KEY) \
  --broadcast \
  --verify \
  --etherscan-api-key $(BASESCAN_API_KEY) \
  --slow \
  -vvvv
```

Supplying secrets this way encourages plaintext handling (e.g., `.env` files, shell history) and exposes the key in process arguments. Storing private keys in plain text represents an operational security risk, as it increases the chance of accidental exposure through version control, misconfigured backups, or compromised developer machines.

A more secure approach is to use Foundry's wallet management features, which allow encrypted key storage. For example, a private key can be imported into a local keystore using `cast`:

```
cast wallet import deployerKey --interactive
```

This key can then be referenced securely during deployment:

```
DEPLOYER_ACCOUNT ?= deployerKey
DEPLOYER_SENDER  ?= $(shell cast wallet address $(DEPLOYER_ACCOUNT) 2>/dev/null)

forge script script/Deployer.s.sol:Deployer \
  --rpc-url $(BASE_SEPOLIA_RPC) \
  --account $(DEPLOYER_ACCOUNT) \
  --sender $(DEPLOYER_SENDER) \
  --broadcast \
  --verify \
  --etherscan-api-key $(BASESCAN_API_KEY) \
  -vvvv
```

and

```
cast wallet list | grep -q "$(DEPLOYER_ACCOUNT)" || { \
  echo "$(RED)ERROR: Keystore account '$(DEPLOYER_ACCOUNT)' not found$(NC)"; exit 1; }
```

For additional guidance, see this explanation video by Patrick.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.1.3 Dust amounts of rewards get stuck in the Vault

Description: In `StakingVault`, rewards are distributed through the `notifyRewardAmount` function. However, due to the current implementation, some rewards can remain stuck in the contract because of rounding errors.

For example:

- User1 stakes `1e18`
- User2 stakes `7e18`
- The total reward amount is `1e18`

Because of rounding, the actual distributed reward will be less than `1e18`, specifically `999999999999000000`. This leaves a small portion of the reward undistributed and permanently stuck in the contract.

```
function notifyRewardAmount(uint256 ilvAmount)
        external
        override
        nonReentrant
        whenNotPaused
        onlyRole(DISTRIBUTOR_ROLE)
    {
        // If no rewards are provided, return
        if (ilvAmount == 0) return;

        // Cache total staked for gas efficiency
        uint256 _totalStaked = totalStaked;

        // When no one is staking, return (prevents division by zero)
        if (_totalStaked == 0) {
            return;
        }

        // Update rewards-per-share accumulator
        accIlvPerShare += (ilvAmount * Constants.ACC_PRECISION) / _totalStaked;

        emit RewardsNotified(ilvAmount, accIlvPerShare);
    }
```

Impact: A portion of the reward tokens will remain stuck in the contract and will never be claimable by users, leading to inefficient distribution of rewards.

Proof of Concept: The following test demonstrates the issue and can be added to `StakingVault.t.sol`:

```
function test_twoUsersWhoStakeAreEligibleForAllRewards() public {
    address user = makeAddr("User1");
    address user2 = makeAddr("User2");
    uint96 amount = 1e18;
    uint32 duration = 31 days;
    uint96 notifyAmt = 1e18;

    // User 1 and User 2 deposit
    approveAndDeposit(user, amount, duration);
    approveAndDeposit(user2, amount * 6, duration);

    // Fund and notify rewards
    ilv.mint(address(vault), notifyAmt);
    vm.prank(admin);
    vault.notifyRewardAmount(notifyAmt);

    // Check if total pending rewards equal notifyAmt
    uint256 pending = vault.pendingRewards(user);
    uint256 pending2 = vault.pendingRewards(user2);
    assertEq(pending + pending2, notifyAmt);
}
```

The assertion will fail because the total pending rewards do not equal the notified amount.

Recommended Mitigation: Increase the precision of the reward calculation by changing `ACC_PRECISION` from `1e12` to `1e18`. This adjustment reduces the rounding error to at most **1 wei**, ensuring nearly all rewards are distributed correctly.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.1.4 Distributor can over-allocate claimables, causing temporary claim DoS

Description: `L2RevenueDistributorV3::distribute` credits claimable pools by incrementing `pendingIlv` but does not reserve tokens for those future claims. Subsequent distributions that transfer ILV out to vaults can consume the same on-chain balance the claimables depend on. This can leave `sum(pendingIlv)` greater than the contract's ILV balance, so a later `L2RevenueDistributorV3::claimPool` reverts until the distributor is topped up.

Impact: The system relies on ops keeping the distributor funded; if they don't, claimable pools can be temporarily DoS'd until a top-up. No permanent loss of funds, but availability is affected and accounting becomes misleading.

Proof of Concept: Add this test to `L2RevenueDistributorV3.t.sol`:

```solidity
    /// PoC: Overcommit claimables via two distributions -> claim reverts (insufficient balance)
    function test_PoC_Overcommit_BricksClaimables_TwoDistributes() public {
        // 1) Seed distributor
        uint256 amount1 = 1_000e18; // clean number to avoid rounding noise
        ilv.mint(address(distributor), amount1);

        // 2) First distribution. Pool 2 (claimable, 2000 bps) accrues 20% pending; 80% sent to vaults.
        vm.prank(operator);
        distributor.distribute(amount1);

        uint256 expectedPendingAfter1 = (amount1 * 2000) / Constants.BPS_DENOMINATOR;
        assertEq(pendingOf(2), expectedPendingAfter1, "pending after first distribute");
        assertEq(ilv.balanceOf(address(distributor)), expectedPendingAfter1, "residual ILV equals
        ↪  claimable pending");

        // 3) Naive second distribution using the entire remaining balance.
        uint256 amount2 = ilv.balanceOf(address(distributor)); // == expectedPendingAfter1
        vm.prank(operator);
        distributor.distribute(amount2);

        // Now: balance < pending (overcommitted)
        uint256 pendingNow = pendingOf(2); // 20% of amount1 + 20% of amount2
        uint256 balNow = ilv.balanceOf(address(distributor));
        assertLt(balNow, pendingNow, "distributor balance < claimable pending");

        vm.prank(admin);
        distributor.setClaimAllowlist(2, address(this), true);

        // 4) Claims are bricked until top-up
        vm.expectRevert("balance"); // SafeERC20 transfer fails due to insufficient ILV
        distributor.claimPool(2, address(this));
    }
```

Recommended Mitigation: Track what's already promised to claimables and ensure distributions only spend the unreserved balance:

- Add a global accumulator:

```solidity
    uint256 public reservedClaimables;
```

- When allocating to a claimable pool, increase the reserve; when claiming, decrease it:

```solidity
    function _applyAllocation(Pool storage pool, uint256 amount) internal {
        if (pool.kind == PoolKind.Vault) {
            ilv.safeTransfer(pool.recipient, amount);
            IStakingVaultMinimal(pool.recipient).notifyRewardAmount(amount);
        } else {
            pool.pendingIlv += amount;
            reservedClaimables += amount;
        }
    }
```

```
function claimPool(uint256 id, address to) external nonReentrant whenNotPaused {
    // ...existing checks...
    uint256 amount = pool.pendingIlv;
    if (amount == 0) revert NothingToClaim(id);
    pool.pendingIlv = 0;
    reservedClaimables -= amount;      // reduce reserved on successful claim
    ilv.safeTransfer(to, amount);
    emit PoolClaimed(id, to, amount);
}
```

- Before performing a distribution, ensure the request does not exceed the unreserved ILV on the contract:

```
error InsufficientUnreservedBalance();

function distribute(uint256 ilvAmount) external nonReentrant whenNotPaused
↪   onlyRole(OPERATOR_ROLE) {
    if (ilvAmount == 0) revert DistributeAmountZero();

    if (ilvAmount > ilv.balanceOf(address(this) - reservedClaimables) revert
    ↪   InsufficientUnreservedBalance();

    // proceed with existing allocation logic...
}
```

This enforces `balance - claimable >= amount` at the start of `distribute()`, guaranteeing that claimable promises are always backed by on-contract tokens and preventing overcommit/DoS of `claimPool()`.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.1.5 Reserve-weighted allocations use spot reserves

Finding (Low): Reserve-weighted allocation uses spot reserves → price-manipulable

Description: `L2RevenueDistributorV3::_computeReserveUnits` derives LP "units" from the current pair reserves (`getReserves()` and `totalSupply()`), i.e., a spot snapshot:

```
IAerodromePair pair = IAerodromePair(pairAddr);
(uint256 r0, uint256 r1,) = pair.getReserves();
address t0 = pair.token0();
address t1 = pair.token1();
uint256 ilvReserve = t0 == address(ilv) ? r0 : (t1 == address(ilv) ? r1 : 0);
```

This allows for manipulation of the reserves.

Impact: A caller can briefly skew LP reserves right before `distribute` to steer that call's rewards toward a chosen LP vault. The cost is a temporary trade + fees; repeating the tactic can compound gains. Base's lack of a public mempool reduces public MEV but doesn't eliminate manipulation completely.

Recommended Mitigation: Consider using a TWAP of reserves over a fixed window from the pair's `observations`. Revert if the oracle history cannot satisfy the requested window. Sufficient observation length can be guaranteed by the protocol when deploying the pool.

```
// Extend the pair interface as needed for your Aerodrome build.
interface IAerodromePairOracle is IAerodromePair {
    function observationLength() external view returns (uint256);
    function lastObservation()
        external
        view
        returns (uint256 timestamp, uint256 reserve0Cumulative, uint256 reserve1Cumulative);
    function observations(uint256 index)
        external
        view
        returns (uint256 timestamp, uint256 reserve0Cumulative, uint256 reserve1Cumulative);
}

library OracleLib {
    error OracleInsufficientHistory(); // fewer than 2 observations
    error OracleWindowNotSatisfied(uint256 availableWindow, uint256 requiredWindow);

    /// @dev Returns average reserves over `windowSecs` using cumulative reserve observations.
    function averageReserves(address pairAddr, uint32 windowSecs)
        internal
        view
        returns (uint256 avgR0, uint256 avgR1)
    {
        IAerodromePairOracle pair = IAerodromePairOracle(pairAddr);
        uint256 len = pair.observationLength();
        if (len < 2) revert OracleInsufficientHistory();

        (uint256 tNew, uint256 r0New, uint256 r1New) = pair.lastObservation();
        uint256 tTarget = tNew - uint256(windowSecs);

        // Walk back to an observation at or before tTarget (use binary search if ring buffer is large).
        uint256 idx = len - 2;
        (uint256 tOld, uint256 r0Old, uint256 r1Old) = pair.observations(idx);
        while (tOld > tTarget) {
            if (idx == 0) {
                revert OracleWindowNotSatisfied(tNew - tOld, windowSecs);
            }
            idx--;
            (tOld, r0Old, r1Old) = pair.observations(idx);
        }

        uint256 dt = tNew - tOld;
        if (dt == 0) revert OracleWindowNotSatisfied(0, windowSecs);

        avgR0 = (r0New - r0Old) / dt;
        avgR1 = (r1New - r1Old) / dt;
    }
}

contract L2RevenueDistributorV3 {
    // ...
    uint32 internal constant TWAP_WINDOW = 15 minutes;

    function _computeReserveUnits(Pool storage pool) internal view returns (uint256) {
        if (pool.kind != PoolKind.Vault) return 0;

        // ILV single-asset vault unchanged
        if (pool.underlyingToken == address(ilv)) {
            return IStakingVaultMinimal(pool.recipient).totalStaked();
        }

        // LP vault: use TWAP average reserves; revert if oracle history is insufficient
        address pairAddr = pool.underlyingToken;
        if (pairAddr == address(0)) return 0;

        (uint256 avgR0, uint256 avgR1) = OracleLib.averageReserves(pairAddr, TWAP_WINDOW);

        IAerodromePair p = IAerodromePair(pairAddr);
        address t0 = p.token0();
        address t1 = p.token1();
        uint256 ilvReserveAvg = t0 == address(ilv) ? avgR0 : (t1 == address(ilv) ? avgR1 : 0);
        if (ilvReserveAvg == 0) return 0;
        uint256 lpSupply = p.totalSupply();
        if (lpSupply == 0) return 0;

        uint256 vaultLp = IStakingVaultMinimal(pool.recipient).totalStaked();
        return (ilvReserveAvg * vaultLp) / lpSupply;
    }
}
```

Illuvium: Acknowledged; Reserve weighted pool allocations will stay using spot reserves from Aerodrome pairs. We'll use reasonable slippage protection params to ensure frontrunning isn't profitable/feasible + count with Base's trusted sequencer.

7.1.6 Dust rewards can become permanently locked

Description During `distribute()`, leftover amounts ("dust") can remain as a remainder. These remainders are attempted to be allocated to the first eligible participant. However, if the remainder is **smaller than** `totalStaked / 1e18`, the reward is not distributed and instead gets permanently stuck in the contract.

```
if (rwDistributed < remainder) {
    uint256 dust = remainder - rwDistributed;
    rwDistributed += _allocateFirstEligible(dust, poolCount, units);
}
```

Impact Over time, repeated distributions with such small remainders accumulate, leading to a growing amount of ILV locked in the contract and never reaching stakers.

Recommended Mitigation Apply the threshold check **inside** `_applyAllocation` to ensure that only meaningful amounts are distributed.

```
function _applyAllocation(Pool storage pool, uint256 amount) internal {
    // skip if dust is too small to be distributed
    if (amount * Constants.ACC_PRECISION <= pool.totalStaked) {
        return;
    }

    if (pool.kind == PoolKind.Vault) {
        ilv.safeTransfer(pool.recipient, amount);
        IStakingVaultMinimal(pool.recipient).notifyRewardAmount(amount);
    } else {
        pool.pendingIlv += amount;
    }
}
```

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.1.7 Equal distribution fallback logic error

Description: In the `L2RevenueDistributorV3::_allocateEqual()` function, when the total reserve units are zero, the contract falls back to equal distribution of remaining ILV to active reserve-weighted pools. However, the eligibility check for Vault pools includes the condition `units[i] > 0`, but `units[i]` is always zero in this scenario (since `totalUnits` is zero). This results in Vault pools being incorrectly excluded from the equal distribution, allowing only Claimable pools to receive funds.

Impact: Vault pools may not receive their intended share of ILV distributions when reserve calculations result in zero units, leading to unfair allocation of rewards. This could cause loss of expected yield for users staking in Vault pools, potentially undermining the protocol's incentive mechanisms.

Recommended Mitigation: Modify the eligibility check in `_allocateEqual()` to not depend on `units[i]` for Vault pools when in equal distribution mode. Instead, simply check that the pool is active and reserve-weighted. For example:

```
bool isEligible = (pool.kind == PoolKind.Claimable) || (pool.kind == PoolKind.Vault);
```

This ensures both Claimable and Vault pools are included in the equal distribution when reserves are zero. Additionally, consider adding a comment to clarify this behavior.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2 Informational

7.2.1 Unused `factory` storage in `L2RevenueDistributorV3`

Description: `L2RevenueDistributorV3` stores `IAerodromeFactory public factory` and accepts it in `initialize`, but never reads or uses it. Swaps rely on `routes[i].factory`, and reserve math uses the LP pair address directly; no logic references the `factory` variable.

- If not needed: stop passing it to `initialize` and remove the public variable.
- If intended for safety: enforce it by checking `routes[i].factory == address(factory)` in swap functions to pin swaps to the approved factory.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.2 Lack of events emitted for important state changes

Description: * `StakingVault::setLockDurations`

- `L2RevenueDistributorV3::setClaimAllowlist`

Lack of events leads to reduced on-chain transparency and weaker off-chain monitoring. Indexers/alerting systems cannot detect parameter changes, making audits, incident response, and UX debugging harder.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.3 Redundant recomputation of accrued rewards in `StakingVault::_updateRewards`

Description: In `StakingVault::_updateRewards`, `accrued` is calculated and then the exact same expression is recomputed for `totalRewardDebt`. The second calculation is unnecessary.

```
uint256 accrued = (staked * _acc) / _scale;
if (accrued > userInfo.totalRewardDebt) {
    userInfo.storedPendingRewards += accrued - userInfo.totalRewardDebt;
}
// @audit Recomputed unnecessarily
userInfo.totalRewardDebt = (staked * _acc) / _scale;
```

Consider reusing the already computed value:

```
uint256 accrued = (staked * _acc) / _scale;
if (accrued > userInfo.totalRewardDebt) {
    userInfo.storedPendingRewards += accrued - userInfo.totalRewardDebt;
}
userInfo.totalRewardDebt = accrued;
```

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.4 Consider adding a minimum deposit amount in `StakingVault`

Description: `StakingVault.deposit` accepts any positive `amount`, including 1 wei. Extremely small stakes don't make economic sense and can amplify integer-rounding edge cases in the rewards-per-share model. These style of deposits are only used by black hats to manipulate vaults.

Consider adding a configurable minimum deposit threshold and enforce it in `deposit`:

```
error DepositAmountTooSmall(uint256 amount, uint256 minDeposit);

uint256 public minDeposit; // set by admin (e.g., during initialize), adjustable via setter + event

function setMinDeposit(uint256 newMin) external onlyRole(ADMIN_ROLE) {
    minDeposit = newMin;
    emit MinDepositUpdated(newMin);
}

function deposit(uint256 amount, uint64 lockStakeDuration) external whenNotPaused nonReentrant returns
↪   (uint256) {
    if (amount < minDeposit) revert DepositAmountTooSmall(amount, minDeposit);
    // ...existing logic...
}
```

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.5 Missing sanity check for reasonable duration values

Description: `StakingVault::initialize` and `StakingVaultFactory::deployVault` fail to validate that lock durations are reasonable (not zero, not extremely large).

Admin could accidentally set `minLockDuration = 0` or both `minLockDuration` and `maxLockDuration` to extremely high values as long as `minLockDuration` is less or equal to `maxLockDuration`.

Recommended Mitigation: Add a sanity check to prevent durations that are not reasonable. Use constants to determine an acceptable range of values.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.6 Missing `address(0)` check in `setDistributor`

Description: The `StakingVault::setDistributor` function allows administrators to grant or revoke the DISTRIBUTOR_ROLE to specified accounts, but it lacks validation to prevent assigning this critical role to the zero address (address(0)). While this doesn't create an immediate security vulnerability, it represents an operational risk that could temporarily disrupt the reward distribution mechanism if this is the only address possessing the given role.

Recommended Mitigation: Add a zero-address validation check at the beginning of the function:

```
function setDistributor(address account, bool enabled) external onlyRole(ADMIN_ROLE) {
    if (account == address(0)) revert AddressZero("account");

    if (enabled) {
        _grantRole(DISTRIBUTOR_ROLE, account);
    } else {
        _revokeRole(DISTRIBUTOR_ROLE, account);
    }
}
```

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.7 Lack of validation of `Pool.underlyingToken == StakingVault.stakingAsset`

Description: `L2RevenueDistributorV3::_computeReserveUnits` decides whether a Vault pool is an ILV vault or an LP vault based on `pool.underlyingToken`. However, there is no check that this value actually matches the vault's true staking asset (`StakingVault.stakingAsset`). A mismatch (by config error or upgrade) would cause the distributor to compute reserve units against the wrong token/pair, skewing reserve-weighted rewards.

Consider validating on pool creation/update that, for `PoolKind.Vault`, `pool.underlyingToken == StakingVault(recipient).stakingAsset()`. Alternatively, remove `underlyingToken` from config for Vault pools and always read the asset from the vault.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.

7.2.8 Inconsistent `recipient` handling for claimable pools

Description: `L2RevenueDistributorV3.Pool.recipient` is documented as "vault address or claimable recipient key":

```
address recipient; // vault address or claimable recipient key
```

However, `L2RevenueDistributorV3::claimPool(id, to)` ignores it and transfers ILV to the caller-supplied `to`. Meanwhile, the deploy script sets a dummy `pool.recipient = address(0x1111)` for the claimable pool, while allowlisting `helperConfig.getL1PoolAdmin()` elsewhere. This mismatch is confusing and brittle if future logic starts using `recipient`.

Consider either clarifying the documentation that `recipient` is not used for claimable pools or change `claimPool` to send to `recipient` and remove the placeholder in the deploy script.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified; The documentation regarding recipient was made more clear.

7.3 Gas Optimization

7.3.1 Unoptimised overflow check in `_assertFixedBpsCap`

Description: In the `_assertFixedBpsCap()` function, the overflow check `if (totalBps > Constants.BPS_DENOMINATOR)` is performed inside the loop for each pool iteration. This causes unnecessary conditional evaluations on every iteration, increasing gas usage. The check only needs to be performed once after summing all relevant `bps` values.

Recommended Mitigation: Accumulate `totalBps` inside the loop, then perform the overflow check once after the loop:

```
function _assertFixedBpsCap() internal view {
    uint256 totalBps;
    for (uint256 i; i < pools.length; i++) {
        Pool storage pool = pools[i];
        if (!pool.active || pool.mode != PoolMode.FixedBps) continue;
        totalBps += pool.bps;
    }
    if (totalBps > Constants.BPS_DENOMINATOR) {
        revert FixedBpsOverflow(totalBps);
    }
}
```

This reduces gas cost by eliminating unnecessary conditional checks inside the loop.

Illuvium: Fixed in commit 5f273bc.

Cyfrin: Verified.