Pashov Audit Group

Lodestar
AI Security Review

August 20th 2026

Executive Summary

A security review of Lodestar was conducted entirely by Pashov Audit Group's AI auditing system. This report documents 7 issues across the reviewed codebase.

Project	Lodestar
Repository	github.com/lodestarprotocol/lodestar - main - c1bd24a

SCOPE

Reviewed commit: `c1bd24a` on `main` - the last commit touching `src/` ; `git diff c1bd24a..main -- src/` is empty and the scope is frozen for the engagement.

In scope (1,199 nSLOC / 2,146 lines incl. comments):

- `src/LodestarLoanBook.sol` - 835
- `src/LodestarPool.sol` - 161
- `src/LodestarOracle.sol` - 133
- `src/flare/*.sol` and `src/interfaces/*.sol` - 70

Out of scope: `web/` , `webapp/` , `script/` , `test/` , and the off-repo keeper.

Build: Solidity 0.8.28, OpenZeppelin 5.1.0, Foundry (optimizer 200, `via_ir = true`).

Tests: 373 unit tests plus 27 fork tests (Flare + Coston2), all green.

About

Pashov Audit Group is a security research collective focused on smart-contract and protocol security. This report documents the issues surfaced during a security review conducted entirely by Pashov Audit Group's AI auditing system.

METHODOLOGY

This security review was conducted entirely by Pashov Audit Group's AI auditing system, which reviewed the in-scope code and reported the issues it found. Each issue was then triaged and validated, with duplicates merged and severities finalized. Every accepted issue appears below with its impact and a recommended remediation. The client's response to each issue was then reviewed, and its resolution status - fixed, acknowledged, or invalidated - is recorded with the finding.

DISCLAIMER

A security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where our AI auditing system tries to find as many issues as possible. We can not guarantee 100% security after the review. Subsequent reviews, bug bounty programs and on-chain monitoring are strongly recommended. The client's remediation of each issue was reviewed after the audit; the outcome is recorded in the finding's Resolution section.

Risk Classification

Critical	Loss or compromise of critical assets, or protocol-wide impact. Must be fixed before launch.
High	Significant material loss of assets, or serious harm to a group of users.
Medium	Moderate material loss, or moderate harm, often under conditional assumptions.
Low	Minor impact, limited scope, or requires unlikely conditions to trigger.
Info	No direct security impact - best-practice, code-quality or hardening suggestion.

Findings

SEVERITY	COUNT
Medium	5
Low	2
Total	**7**

ID	TITLE	SEVERITY	STATUS
M-01	Changing an LST rate provider retroactively invalidates openRate snapshots	Medium	Fixed
M-02	Gross impairment can orphan reserve-backed recovery after lender exits	Medium	Fixed
M-03	Loan origination bypasses the impairment checkpoint used by utilization limits	Medium	Acknowledged
M-04	Reserve losses do not reduce the contributed-capital floor	Medium	Fixed
M-05	Withdrawals can consume cash backing unvested fees	Medium	Fixed
L-01	Haircuted oracle price can reject honest settlement proceeds	Low	Fixed
L-02	previewRedeem can overstate proceeds before mandatory impairment synchronization	Low	Fixed

Medium Severity

[M-01] Changing an LST rate provider retroactively invalidates openRate snapshots

Medium Fixed

LOCATION

`LodestarOracle:setFeed:92-100`

SUMMARY

When a loan opens, the collateral's current rate is snapshotted into `openRate` so that only appreciation after opening is subject to the configured yield skim. Repayment and partial-release then read the live rate via `oracle.rateOf` and compare it against that stored `openRate` .

The problem is that `setFeed` can swap the collateral's rate provider while loans are still open, without touching their existing `openRate` snapshots. After a swap, the live rate comes from the new provider while the snapshot still reflects the old provider, so the two values are no longer comparable. Clearing the oracle's rate clamp does not repair the loan-level snapshot.

If the old provider reported `1.05e18` at open and a new provider reports `1.20e18` on a different accounting basis, `_returnCollateralPortion` treats the full `1.20e18 - 1.05e18` as post-opening yield even though the difference may just be a change of basis. The same mechanism can also suppress legitimate appreciation in the opposite direction. This is a normal administrative configuration operation, not an attacker-only setup.

IMPACT

After a rate-provider migration, collateral appreciation on already-open loans can be over- or under-counted.

Depending on the relationship between the old and new provider rates, borrowers may lose excess collateral to the reserve, or the reserve may receive less than its configured skim.

EXPLOIT PATH

1. A borrower opens an sFLR loan while provider A reports `1.05e18` , which is stored in `openRate` .
2. An admin replaces the feed's rate provider with provider B, which reports `1.20e18` on a different basis.
3. The loan is repaid or partially repaid; `_returnCollateralPortion` computes gain from `1.20e18 - 1.05e18` .
4. The reserve receives an incorrect skim and the borrower receives an incorrect net collateral amount.

PREREQUISITES

Requires the admin/trusted role to change a feed's rate provider via `setFeed` while loans against that token are still open. No attacker setup is needed; a routine migration triggers it.

RECOMMENDATION

Block provider changes while active loans exist for the token:

```
if (activeLoanCountForToken[token] != 0 && feeds[token].rateProvider != rateProvider) {
revert ActiveLoans();
}
```

Alternatively, version each feed and snapshot that version in every loan, then skip or specially migrate the skim whenever a loan's provider version differs from the current feed version.

RESOLUTION - FIXED

Fixed in commit `f3e83c5` - the second pass, after the initial partial fix.

[M-02] Gross impairment can orphan reserve-backed recovery after lender exits

Medium Fixed

LOCATION

`LodestarLoanBook:_markLoanRaise:1718-1734`

SUMMARY

Impairment is meant to reduce lender NAV only for the portion of an active loan not expected to be recovered, while preserving a coherent ownership claim over any later repayment or first-loss reserve coverage. Reserve capital is held separately in the LoanBook and applied only at settlement.

The implementation instead marks the full gross collateral shortfall against pool NAV. `totalAssets()` subtracts `impairedLoss` but has no corresponding asset or claim for `reserveBalance`. Withdrawals only require fresh impairment synchronization and sufficient idle liquidity, so all remaining shares can be redeemed while the impaired loan is still open.

When settlement later calls `_clearImpairment()` and `_distribute()`, the pool receives the recovery after the shares that absorbed the impairment may already have been burned, leaving that recovery with no lender ownership behind it.

IMPACT

Reserve-backed recovery can arrive after all lender shares have been redeemed, leaving recovered assets in the pool with no ownership record. This can trap or misallocate protocol funds in a shareless pool.

EXPLOIT PATH

1. A collateral crash causes `impair(id)` to mark an active loan's expected loss up to roughly its full principal.
2. `LodestarPool.totalAssets()` falls by that impairment, while the LoanBook's separate reserve stays available but unrepresented in pool NAV.
3. Lenders call `withdraw` or `redeem`; synchronization succeeds and idle liquidity is sufficient, so their shares are burned.
4. A later `buyout` or `settleSwap` clears the impairment and transfers repayment or reserve coverage to the pool, but the exited lenders hold no shares representing that recovery.

RECOMMENDATION

Include the reserve claim in the pool's impairment accounting before allowing exits, limited by the reserve actually available for the loan:

```
uint256 reserveClaim = loanBook.reserveCoverage(id, newLoss);
uint256 netLoss = newLoss > reserveClaim ? newLoss - reserveClaim : 0;
```

Alternatively, prohibit full share exit while impaired loans with unclaimed reserve-backed recovery remain outstanding, or maintain a separate recovery-credit ledger that follows the lender shares and is settled when the loan closes.

RESOLUTION - FIXED

Fixed by the client.

[M-03] Loan origination bypasses the impairment checkpoint used by utilization limits

Medium | Acknowledged

LOCATION

`LodestarPool:disburse:167-176`

SUMMARY

The utilization ceiling is meant to cap new lending against the pool's current risk-adjusted assets, including losses already observable from the live oracle. New loan origination bypasses this because it never refreshes impairment before the check.

`_open` calls `pool.disburse` after pricing only the new collateral. `disburse` reads `totalAssets()`, but that value subtracts only the stored `impairedLoss` and does not derive or refresh losses from current oracle prices. An existing underwater loan therefore stays fully counted until a separate permissionless synchronization runs.

During that stale-accounting window, a new borrower can be funded against an overstated denominator. Once impairment is later synchronized, utilization rises against the now-lower asset base, showing the origination did not enforce the configured limit against current state.

IMPACT

The max-utilization control can be bypassed in the interval between a collateral price deterioration and the next impairment synchronization.

Additional stable is lent while the pool's risk-adjusted asset base is overstated, leaving the pool more utilized and less liquid than its configured risk limit permits.

EXPLOIT PATH

1. Loan A is active and healthy, with `pool.impairedLoss` recorded as zero.
2. Loan A's collateral price falls enough that `_markLoanRaise` would record a loss, but no impairment synchronization has run yet.
3. A borrower opens loan B; `_open` calls `pool.disburse` without synchronizing loan A, so utilization is checked against a stale-high `totalAssets()`.
4. A later `impair` or exit sweep marks loan A's loss, leaving the pool above the configured utilization ratio after loan B has already drained additional idle liquidity.

RECOMMENDATION

Refresh impairment before the utilization check, preferably via a dedicated LoanBook-to-Pool synchronization hook:

```
function disburse(address to, uint256 net, uint256 principal)
external
onlyLoanBook
{
ILoanBookSync(msg.sender).syncImpairment();
uint256 ta = totalAssets();
if ((principalOut + principal) * 10_000 > ta * maxUtilizationBps)
revert OverUtilized();
// update principalOut and transfer funds
}
```

If strict oracle availability is required for risk checks, use the strict synchronization variant and revert origination when the active book cannot be freshly priced, rather than allowing a stale utilization decision.

RESOLUTION - ACKNOWLEDGED

Acknowledged. The finding is unreachable once `totalAssets` clears `sum(exposureCaps) * 10000 / maxUtilizationBps`, which holds at the launch caps; that bound is pinned in CI so it cannot reopen silently.

[M-04] Reserve losses do not reduce the contributed-capital floor

Medium Fixed

LOCATION

`LodestarLoanBook:_distribute:1238-1248`

SUMMARY

The reserve floor is meant to track the remaining externally contributed first-loss capital, while fee and penalty income above that floor should stay withdrawable protocol surplus. When a settlement shortfall consumes reserve funds, the contributed portion should shrink before any protocol surplus is touched.

Instead, the settlement waterfall only decreases the aggregate balance and clamps the floor down only if it now exceeds that balance:

```
reserveBalance -= cover;
if (reserveFloor > reserveBalance) reserveFloor = reserveBalance;
```

With a balance of 200, a floor of 100, and a 50-unit loss, this yields a balance of 150 and a floor still at 100. The floor should have fallen to 50, so 50 units of surplus are stranded behind an overstated floor.

This happens during normal settlement whenever protocol surplus sits above contributed reserve capital; no attacker is required.

IMPACT

Settlement losses permanently overstate the reserve capital protected from owner withdrawal, locking protocol-owned surplus and leaving reserve accounting inconsistent with the documented first-loss waterfall. Legitimate `withdrawReserve` calls revert.

EXPLOIT PATH

1. Call `fundReserve(100)`, then accrue 100 units of fee or penalty income so `reserveBalance = 200` and `reserveFloor = 100`.
2. Settle a defaulted loan whose waterfall has a 50-unit uncovered principal gap.
3. `_distribute` transfers 50 reserve units to the pool and leaves `reserveFloor` at 100.
4. The owner calls `withdrawReserve(100)`, which reverts because the code requires 100 units to remain protected instead of 50.

RECOMMENDATION

Decrease the floor by the reserve loss consumed:

```
reserveBalance -= cover;
reserveFloor = reserveFloor > cover ? reserveFloor - cover : 0;
```

If the protocol intends a different loss-priority model, maintain separate contributed and protocol-surplus balances and update both explicitly during `_distribute`.

RESOLUTION - FIXED

Fixed by the client.

[M-05] Withdrawals can consume cash backing unvested fees

Medium Fixed

LOCATION

`LodestarPool:totalAssets:111-126; LodestarPool:withdraw:227-236; LodestarPool:redeem:238-247`

SUMMARY

The pool holds fee income that is intentionally excluded from lender asset value until it vests. The stable cash backing `lockedFee` must be preserved until the vesting period elapses, since exits should not consume the cash supporting that deferred accounting.

The withdrawal guard uses the full token balance through `available()` and does not reserve `unvestedFee()`. A permitted withdrawal can therefore leave `balanceOf(pool) < unvestedFee()`. The next `totalAssets()` call clamps the fee deduction to the remaining balance:

```
uint256 uf = unvestedFee();
if (uf > bal) uf = bal;
if (uf >= net) return net;
return net - uf;
```

Because the clamp shrinks the deduction, the missing portion of the fee appears earned before it has vested, corrupting share accounting.

IMPACT

Remaining lenders and later entrants receive distorted share pricing: unvested fee income is treated as earned once its idle stable backing has been withdrawn. This transfers deferred fee value into current share valuation and can be induced through ordinary withdrawals.

EXPLOIT PATH

1. A pool with 300 idle stable and 800 principalOut receives a 100-unit fee recorded as unvested via `lockFee(100)` .
2. Before vesting completes, a lender withdraws 300 units; the check passes because `assets ⩽ available()` returns the full 300.
3. The pool balance becomes zero while `unvestedFee()` remains 100.
4. `totalAssets()` computes `net = 800` , clamps `uf` to zero, and reports 800 instead of 700, shifting the deferred fee into the remaining share valuation.

RECOMMENDATION

Reserve the outstanding fee lock from exit liquidity:

```
function available() public view returns (uint256) {
uint256 bal = IERC20(asset()).balanceOf(address(this));
uint256 uf = unvestedFee();
return bal > uf ? bal - uf : 0;
}
```

Alternatively, enforce the same invariant directly in `withdraw()` and `redeem()` by requiring the post-withdrawal balance to remain at least `unvestedFee()` . The shared `available()` change is less error-prone because both exit paths already use the liquidity helper.

RESOLUTION - FIXED

Fixed by the client.

Low Severity

[L-01] Haircuted oracle price can reject honest settlement proceeds

Low Fixed

LOCATION

`LodestarLoanBook:settleSwap:1237-1244; LodestarOracle:priceUsd18:212-219`

SUMMARY

The proceeds sanity check in `settleSwap()` is meant to reject implausibly large stable balance deltas while allowing an honest sale at roughly market value. A collateral haircut is a risk-management discount for underwriting and recovery math; it does not mean the collateral cannot sell for its full unhaircuted market value.

The oracle applies the haircut before returning `priceUsd18`, and the loan book caches that discounted result in `lastPrice18`. `settleSwap()` then multiplies this already-haircuted price by 1.5 to form `saneMax`, so the haircut is effectively applied a second time to the market-value ceiling.

With a 50% permitted haircut, a normal sale at true market value exceeds the resulting ceiling (75% of market) and reverts with `ProceedsTooHigh`.

IMPACT

Legitimate DEX settlements revert with `ProceedsTooHigh` when collateral carries a substantial conservative haircut, even under valid configuration.

This forces users toward the separate buyout path or delays default resolution, reducing settlement liveness on the router path.

EXPLOIT PATH

1. Configure a collateral feed with `haircutBps = 5000`, where the unhaircuted market value of the sale amount is 1,000 USDT.
2. The oracle returns and caches a haircuted value of 500 USDT.
3. `settleSwap()` computes `saneMax = 500 * 150% = 750 USDT`.
4. The whitelisted router honestly returns 1,000 USDT, but the call reverts because proceeds exceed the haircuted ceiling.

RECOMMENDATION

Expose an unhaircuted market-price value for the sanity check while keeping the haircuted value for risk accounting.

```
function marketPriceUsd18(address token) external view returns (uint256) {
return _rawPriceUsd18(token);
}
```

```
uint256 p18 = oracle.marketPriceUsd18(L.collateral);
uint256 saneMax =
(_usd18ToStable((p18 * toSell) / _unit(L.collateral)) * 15_000) / 10_000;
if (proceeds > saneMax) revert ProceedsTooHigh();
```

RESOLUTION - FIXED

Fixed in commit `f3e83c5` - the second pass, after the initial partial fix.

[L-02] previewRedeem can overstate proceeds before mandatory impairment synchronization

Low Fixed

LOCATION

`LodestarPool:previewRedeem/redeem:132-158, 250-255`

SUMMARY

`previewRedeem` is inherited unchanged from ERC4626, so it reads `totalAssets()` without synchronizing the loan book. The `redeem()` path, however, calls `syncImpairmentForExit()` first, which can raise `impairedLoss` and reduce `totalAssets()` before the redemption math runs.

As a result, a quote taken from `previewRedeem` immediately before redemption uses the higher pre-synchronization `totalAssets()`, while the actual redemption prices shares against the reduced value once a fresh collateral crash is marked. The preview can therefore report more assets than the redemption returns. This produces stale quotes and failed or mis-sized exits rather than direct unauthorized value extraction.

IMPACT

Lenders, routers, and integrators can receive a redemption quote higher than the assets actually returned after a fresh impairment sweep.

Transactions that depend on the preview for minimum-output checks or downstream accounting can revert or operate on materially stale assumptions during a collateral crash.

EXPLOIT PATH

1. A pool holds active loans whose collateral has fallen below recovery value, but `impairedLoss` has not yet been raised.
2. A lender calls `previewRedeem(shares)`, which converts using the higher pre-synchronization `totalAssets()`.
3. The lender calls `redeem()` while the oracle is live; `syncImpairmentForExit()` raises `impairedLoss` for the underwater loans.
4. `super.redeem()` prices the shares against the reduced `totalAssets()`, returning fewer stable assets than the earlier preview reported.

RECOMMENDATION

Provide a view-only preview that applies the same impairment model to the current active loans, or expose a synchronized quote path whose result is used by `redeem`. This requires the loan book to offer a view-only `previewImpairment()` that reproduces `_markLoanRaise` without storage writes.

```
function previewRedeem(uint256 shares) public view override returns (uint256) {
uint256 loss = loanBook.previewImpairment();
uint256 assets = totalAssets() - loss;
return _convertToAssets(shares, Math.Rounding.Floor, assets);
}
```

If that simulation is too expensive, provide an explicit synchronized quote transaction or clearly document that integrations must tolerate the mandatory pre-redemption revaluation.

RESOLUTION - FIXED

Fixed in commit `f3e83c5` - the second pass, after the initial partial fix.