SHERLOCK

Security Review For Altura

Collaborative Audit Prepared For: **Altura**
Lead Security Expert(s): **0xadrii**
ParthMandale
Date Audited: **January 27 - January 31, 2026**

Introduction

Altura is a multi-strategy yield protocol deployed on HyperEVM, designed to provide sustainable, risk-adjusted returns through a diversified set of non-directional and asset-backed strategies.

Users deposit USDT into a single vault, and Altura allocates capital across multiple yield sources using an institutional-grade execution and risk framework. Strategy selection, capital allocation, and rebalancing are handled programmatically, allowing users to access professional yield generation without managing trades or exposure themselves.

Altura's design prioritizes capital efficiency, transparency, and long-term sustainability, rather than short-term incentives or speculative returns.

Scope

Repository: AlturaTrade/contracts

Audited Commit: 6471c5848243689c5117cf400c5aeff8c2d2d893

Final Commit: 6471c5848243689c5117cf400c5aeff8c2d2d893

Files:

- contracts/NavOracle.sol
- contracts/NavVault.sol

Final Commit Hash

6471c5848243689c5117cf400c5aeff8c2d2d893

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	4	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: Claim withdrawals can spend assets reserved for exit fees causing loss of funds to protocol [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/3

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`claimWithdrawal` checks only the total vault balance and can pay users with assets that were reserved as `accruedExitFeesAssets`. This breaks fee accounting and can cause later `sweepExitFees` to fail even though fees were accrued, causing loss of funds to the protocol.

Vulnerability Detail

`claimWithdrawal` uses `liquid = asset.balanceOf(vault)` and compares it directly to `full Assets`. Because `liquid` already includes the fee reserve, the function can transfer out assets that were intended to be kept for the protocol's exit fees. Function `moveAssets` explicitly protects fee reserves by adding correct check protection, but `claimWithdrawal` does not, creating inconsistent treatment of the reserved balance.

Impact

Fee reserves can be drained by queued claims, preventing the protocol from sweeping accrued exit fees and leading to accounting inconsistencies and loss of funds.

Code Snippet

function `claimWithdrawal` ->

```
uint256 fullAssets = convertToAssets(sharesEscrow);
if (fullAssets == 0) revert ZeroAmount();
if (liquid < fullAssets) revert InsufficientLiquidity();
```

Tool Used

Manual Review

Recommendation

Exclude `accruedExitFeesAssets` from the liquidity check in `claimWithdrawal` (similar to moveAssets).

Fix:

```
- if (liquid < fullAssets) revert InsufficientLiquidity();
+ if ((liquid - accruedExitFeesAssets) < fullAssets) revert InsufficientLiquidity();
```

Discussion

Parthmandale

High Severity = High Impact * Medium Likelihood

AlturaTrade

Acknowledged but since the liquidity on the smart contract for withdrawals is funded by operations team directly hence its not a high issue

Parthmandale

> Acknowledged but since the liquidity on the smart contract for withdrawals is funded by operations team directly hence its not a high issue

@AlturaTrade obviously liquidity is supposed to be funded by operations team, but at the end of the day adding this a security check is necessary here, as we can clearly see similar checks are used for other functions to avoid similar issue.

AlturaTrade

Acknowledged, but since contract is already deployed we cannot make changes I Acknowledge the issue but suggests that the Severity should be medium rather than High

Issue M-2: Queued claim can revert on insufficient liquidity, forcing a paid exit [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/4

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

If liquidity is low at claim time, `claimWithdrawal` reverts and the user cannot access funds despite waiting the full cooldown `epochSeconds`. The only practical workaround is to `cancelWithdrawal and` withdraw‘ instantly, which applies the exit fee.

Vulnerability Detail

`claimWithdrawal` requires full liquidity to pay the entire `fullAssets` amount; partial claims are not supported. When the vault balance isn't sufficient (e.g., PPS increased but and that much assets liquidity is not available), the claim reverts.

The only workaround for users in this case to get back their asset is they must `cancelWithdrawal` and use the instant `withdrawal` path to get back atleast available asset ie. making a partial withdrawal of their shares, which charges the exit fee, negating the fee□free queued withdrawal they waited for.

Impact

User claim DoS and loss of funds to user: the user either remains stuck(DoS) or pays an exit fee through the instant withdraw method, that too after completing the cooldown `epochSeconds`, resulting in time loss and unexpected cost.

Code Snippet

function claimWithdrawal: Lack of asset liquidity

```
if (liquid < fullAssets) revert InsufficientLiquidity();
```

function withdraw: user needs to pay fees

```
uint256 fee = _feeOnNet(assets);
```

Tool Used

Manual Review

Recommendation

Support partial claims. Alternatively, allow fee-free partial redemption for queued withdrawals when liquidity is insufficient.

Discussion

AlturaTrade

The vault always have 3-5% of the TVL reserved on contract to address this and allow instant withdrawals Just before the epoch closing time the OPERATOR calculates the total amount to be redeemed and transfer the assets to the contract

Parthmandale

> The vault always have 3-5% of the TVL reserved on contract to address this and allow instant withdrawals Just before the epoch closing time the OPERATOR calculates the total amount to be redeemed and transfer the assets to the contract

@AlturaTrade My issues raise concerns for users who are trying to claim but are unable to do so due to a lack of asset liquidity. This can occur due to various edge cases–for example, users may have called queueWithdrawal after the OPERATOR transferred the total required assets, or due to other similar reasons.

In such cases, where a user has completed the epoch cooldown period and needs their assets back at that point in time but is unable to claim them, they are left with no option other than an immediate withdrawal. This results in a fee of up to 2% (200 bps) on the withdrawn assets.

You can review the recommendation provided by me, which addresses and resolves this issue.

> Support partial claims. Alternatively, allow fee-free partial redemption for queued withdrawals when liquidity is insufficient.

Now, in case the smart contract is already deployed, then unfortunately we can't help

AlturaTrade

We acknowledge that a queued withdrawal claim may temporarily fail if on-chain liquid assets are insufficient at the time of claim (e.g., due to liquidity being moved for strategy execution or timing mismatches between new queue requests and liquidity replenishment). This is an availability and UX consideration, not a funds-safety issue. In such cases, user funds are never at risk: shares remain escrowed in the vault, the request is not lost, and the user may either claim once liquidity is restored or cancel the request to retrieve their shares. No principal can be drained or bypassed due to this condition. As an additional operational safeguard, the protocol maintains a baseline on-chain liquidity buffer (typically ~3–5% of total assets) within the vault at all times to absorb normal withdrawal demand and reduce the likelihood of claim delays. To further mitigate edge cases, the protocol is backed by a continuous liquidity monitoring and

alerting system. This system tracks queued withdrawal requirements against on-chain liquid assets and triggers persistent alerts to the operations team whenever available liquidity falls below the level required to satisfy claimable or upcoming withdrawals. Alerts remain active until liquidity is replenished (e.g., via fundLiquidity()), or corrective action is taken. We therefore classify this as a non-critical availability risk with layered operational safeguards, not a security or solvency issue.

Issue M-3: Queued claims allow fee‑free timing of exits at favorable PPS [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/6

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Queued withdrawals are priced at claim time, not at queue time. After the epoch boundary, users can keep requests open indefinitely and claim immediately whenever PPS is most favorable, without paying the exit fee.

Vulnerability Detail

`claimWithdrawal` computes `fullAssets = convertToAssets(sharesEscrow)` at the moment of claim, using the current oracle `PPS`. There is no deadline for `claimWithdrawal`, This means a user can:

- First `Deposit` and then immediately `queueWithdrawal`.
- Wait for claimableAt to pass.
- Monitor PPS and only claim when it is high (or front‑run a PPS drop), paying no exit fee.

Because the request remains open indefinitely after `claimableAt`, the user effectively gains a fee‑free timing option on exits.

Impact

Users can avoid the exit fee and time their withdrawals around favorable PPS changes, giving them an advantage over other users who exit via withdraw/redeem and pay fees.

Code Snippet

function `claimWithdrawal`

```
uint256 fullAssets = convertToAssets(sharesEscrow);
```

Tool Used

Manual Review

Recommendation

Have a claim grace period set in function `queueWithdrawal` for every new ID and in the function `claimWithdrawal` apply an exit fee on request IDs that claim after a grace period.

This will prevent users from applying this malicious technique.

Discussion

AlturaTrade

Acknowledged, the smart contract is already deployed and are immutable, hence we cannot make the change

Issue M-4: Rate-limited oracle updates let users exit at stale inflated PPS during sharp drops [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/7

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`maxPpsMoveBps` caps PPS changes per report, so large market drops must be reflected over multiple oracle updates. During this lag, the vault still prices withdrawals using the stale, higher PPS.

Vulnerability Detail

The `reporter` cannot move PPS down by more than `maxPpsMoveBps` in one function `reportNav` call. If the true `NAV` falls sharply, the oracle must "walk down" PPS across multiple transactions. In that window, users can withdraw at the old PPS by front-running, paying only the exit fee, and avoiding the real loss.

Impact

Users can exit at an inflated PPS and can withdraw more assets than the pool's true value. Those assets come out of the shared vault balance, so less liquidity remains for everyone else. And when PPS is later updated, and claims are attempted, remaining users may also face lower payouts or claim failures.

Code Snippet

function `reportNav`

```
uint256 diff = prev > pps1e18 ? prev - pps1e18 : pps1e18 - prev;
if (diff * 10_000 > prev * maxPpsMoveBps) revert TooLargeMove();
```

Tool Used

Manual Review

Recommendation

The fix is trivial. Removing `maxPpsMoveBps` limit can be one option, but this depends upon the protocol design!

Discussion

AlturaTrade

The objective of the strategy is to give 20-25% yearly returns, the per day move is ~(0.054% - 0.068%), and the exit fee is 0.1% for instant withdrawal

Also maxPpsMoveBps can be set to 0 to bypass max movement

Parthmandale

@AlturaTrade

> the per day move is ~(0.054% - 0.068%), and the exit fee is 0.1% for instant withdrawal

Yes that's the point of issue, user can make immediate withdraw in order to make profit.

> Also maxPpsMoveBps can be set to 0 to bypass max movement

Yes, this can be a good workaround for this issue. I reported this issue because in discored channel, you said "if there is sudden movement" the protocol will call this function multiple times, and from that point of view, I created this issue.

> 5m is not a hard boundary, we are publishing at that interval, but if need be we should be able to publish within minute as well (if there is sudden movement)

AlturaTrade

maxPpsMoveBps is set to 0 and PPS is updated on each 5 min interval to ensure the PPS always remains fresh

Issue L-1: Oracle accepts future timestamps, defeating staleness checks [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/8

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`reportNav` allows a reporter to submit a future `ts`, so the vault's staleness check won't fail until that future time plus `maxAllowedStaleness`.

Vulnerability Detail

`NavVault._readPps1e18` only rejects stale data when `block.timestamp > ts + maxAllowedStaleness`. If the `reporter` submits a NAV with `ts` far in the future, this condition won't be met for a long time, and the vault will treat the NAV as fresh even if it hasn't been updated for the intended interval.

Marking this low-severity risk because the reporter is trusted, but it undermines the intended safety constraint.

Impact

Effective staleness protection can be bypassed, letting an outdated NAV remain valid indefinitely.

Code Snippet

function `reportNav`

```
function reportNav(uint256 pps1e18, uint256 ts) external onlyRole(REPORTER_ROLE)
↪   whenNotPaused {
    if (pps1e18 == 0 || ts == 0) revert ZeroValue();
    if (ts < block.timestamp - _maxOracleStaleness) revert StaleTimestamp();
```

Will not revert in check of this `_readPps1e18` function:

```
function _readPps1e18() internal view returns (uint256 pps1e18) {
    (uint256 pps, uint256 ts) = navOracle.pricePerShare();
    if (pps == 0) revert OracleInvalid();
    if (block.timestamp > ts + maxAllowedStaleness) revert OracleStale();
    if (!navOracle.isValid()) revert OracleInvalid();
```

```
    return pps;
}
```

Tool Used

Manual Review

Recommendation

Reject future timestamps, e.g., `require(ts <= block.timestamp)`

Discussion

AlturaTrade

Acknowledged

Issue L-2: Follow CEI Pattern [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/9

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

In function `withdrawWithCheck`, the `maxShares` slippage check happens after `_withdraw` (interaction), so it violates the “check before interaction” pattern. The same pattern appears in the function `redeemWithCheck`, where _withdraw happens before the `minAssets` slippage check.

Recommendation

It is recommended to use the CEI pattern for smart contracts as it improves safety posture and reduces gas waste if the checks fail.

Issue L-3: `executeOracleUpdate` can be used to bypass pps deviation checks [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-altura-jan-27th/issues/10

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `executeOracleUpdate()` function in NavVault allows switching to a new oracle without validating that the new oracle's price per share (pps) is within acceptable bounds compared to the current oracle's price. This bypasses the maxPpsMoveBps protection mechanism that exists in the NavOracle contract's `reportNav()` function, potentially allowing sudden large price changes.

Vulnerability Detail

The NavOracle contract implements price movement restrictions via `maxPpsMoveBps` to prevent sudden large price changes in `reportNav()`:

```
if (maxPpsMoveBps > 0) {
    uint256 prev = lastNav.pps1e18;
    if (prev > 0) {
        uint256 diff = prev > pps1e18 ? prev - pps1e18 : pps1e18 - prev;
        if (diff * 10_000 > prev * maxPpsMoveBps) revert TooLargeMove();
    }
}
```

However, when updating to a new oracle in NavVault.sol:167-179, there is no validation that the new oracle's current price is within an acceptable range of the old oracle's price:

```
function executeOracleUpdate() external onlyRole(DEFAULT_ADMIN_ROLE) {
    address newOracle = pendingOracle;
    if (newOracle == address(0)) revert BadAddress();
    if (block.timestamp < pendingOracleSetAt + ORACLE_TIMELOCK) revert("Oracle
    ↪  update timelocked");

    address old = address(navOracle);
    navOracle = INavOracle(newOracle);  // Direct switch, no price validation

    pendingOracle = address(0);
    pendingOracleSetAt = 0;

    emit OracleChanged(old, newOracle);
}
```

An example scenario would be:

1. Admin queues new oracle via `queueOracleUpdate()`
2. After 1 day timelock, admin calls `executeOracleUpdate()`
3. The vault immediately switches to the new oracle with potentially drastically different pps
4. All vault calculations (`totalAssets()`, `convertToShares()`, etc.) now use the new price

Impact

Info. While this requires admin privileges and has a 1-day timelock, it creates an avenue to bypass price movement protections.

Code Snippet

https://github.com/sherlock-audit/2026-01-altura-jan-27th/blob/main/contracts/contracts/NavVault.sol#L167

Tool Used

Manual Review

Recommendation

Add price validation when executing oracle updates to ensure the new oracle's price is within acceptable bounds.

Discussion

AlturaTrade

Acknowledged

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.