SHERLOCK

Security Review For Altura

Collaborative Audit Prepared For: **Altura**
Lead Security Expert(s): **0xeix**
ParthMandale
Date Audited: **February 20 - February 21, 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.

Scope

Repository: AlturaTrade/contracts

Audited Commit: fc8f6f909d7abfeb0ac38858f235f62a80fdb1e3

Final Commit: 4d5fa8cd2b1c72ee8b6b94e846ff4cf7d812998b

Files:

- contracts/WithdrawalWrapper.sol

Final Commit Hash

4d5fa8cd2b1c72ee8b6b94e846ff4cf7d812998b

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	0	2

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: Redundant allowance reset pattern by using `forceApprove` twice [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-altura-feb-19th/issues/2

Summary

Calling `forceApprove(spender, 0)` immediately before `forceApprove(spender, amount)` is unnecessary when using OpenZeppelin SafeERC20 `forceApprove`, and slightly increases gas/complexity.

Vulnerability Detail

The `forceApprove` function used in the SafeERC20 library already handles non-standard ERC20 approvals (tokens that require setting allowance to zero before updating) by attempting `approve(amount)` and falling back to `approve(0)` then approve(amount) if needed. Manually calling the following line duplicates the behavior:

https://github.com/sherlock-audit/2026-02-altura-feb-19th/blob/main/contracts/contracts/WithdrawalWrapper.sol#L67-68

```
shares.forceApprove(address(vault), 0);
shares.forceApprove(address(vault), sharesAmount);
```

Impact

Extra external calls and slightly greater gas usage.

Tool Used

Manual Review

Recommendation

Use a single call:

```
-shares.forceApprove(address(vault), 0);
+shares.forceApprove(address(vault), sharesAmount);
```

Discussion

AlturaTrade

Fixed: https://github.com/AlturaTrade/contracts/commit/02ab9f04f4b9ded1eb41672c9e4bbb749c70dc37

Issue L-2: Missing Receiver Assertion in function `claim`and `cancel` Can Cause Silent Relayer Mis-Execution [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-altura-feb-19th/issues/3

Summary

Here `claim(uint256 id)` and `cancel(uint256 id)` in `WithdrawalWrapper.sol` trust only the request ID. If the relayer submits the wrong ID by mistake, the contract will still execute as long as status checks pass, with no on-chain mismatch detection against the relayer's intended receiver.

Vulnerability Detail

In `WithdrawalWrapper.sol` both `claim` and `cancel` load `requests[id]` and proceed if:

```
if (r.receiver == address(0)) revert NotFound();
if (r.receiver != expectedReceiver) revert InvalidID();
```

There is no input parameter for the expected receiver, so the contract cannot detect the relayer's indexing mistakes (wrong ID selected from off-chain queueing systems).

As a result, an incorrect ID can be executed successfully, even when the relayer intended to process a different user request for that `claim` or `cancel`

Impact

This is a general health risk and not a direct external exploit, since only `RELAYER_ROLE` calls these functions.

- Wrong user requests may be processed unintentionally, and withdrawal flow reliability could degrade under relayer mistakes.

Code Snippet

https://github.com/sherlock-audit/2026-02-altura-feb-19th/blob/d1ef37e14f24819f8dd64f79d5d78e00a3109fc4/contracts/contracts/WithdrawalWrapper.sol#L81-L84

https://github.com/sherlock-audit/2026-02-altura-feb-19th/blob/d1ef37e14f24819f8dd64f79d5d78e00a3109fc4/contracts/contracts/WithdrawalWrapper.sol#L99-L102

Tool Used

Manual Review

Recommendation

Add an expected Receiver input to both functions and assert it matches stored request data, and Revert on mismatch invalid Requested ID.

```
- error NotFound();
+ error InvalidID();
```

```
-    function claim(uint256 id) external onlyRole(RELAYER_ROLE) nonReentrant {
+    function claim(uint256 id, address expectedReceiver) external
 ↪  onlyRole(RELAYER_ROLE) nonReentrant {
        Request storage r = requests[id];
-        if (r.receiver == address(0)) revert NotFound();
+        if (r.receiver != expectedReceiver) revert InvalidID();
         if (r.status != Status.QUEUED) revert BadStatus();
```

```
-    function cancel(uint256 id) external onlyRole(RELAYER_ROLE) nonReentrant {
+    function cancel(uint256 id, address expectedReceiver) external
 ↪  onlyRole(RELAYER_ROLE) nonReentrant {
        Request storage r = requests[id];
-        if (r.receiver == address(0)) revert NotFound();
+        if (r.receiver != expectedReceiver) revert InvalidID();
         if (r.status != Status.QUEUED) revert BadStatus();
```

Discussion

AlturaTrade

Both claim(id) and cancel(id) strictly transfer assets or shares to requests[id].receiver. There is no parameter in either function that allows the relayer (or any other caller) to override or modify the receiver at execution time.

This is intentional so just by requestId we can process a transaction safely.

Parth0x108

> There is no parameter in either function that allows the relayer (or any other caller) to override or modify the receiver at execution time.

Hi @AlturaTrade, by reading your comment, I think there is a misunderstanding. The issue is not that the caller can override or modify the receiver at execution time.

The issue is operational safety: the relayer can accidentally submit the wrong id (from a queue mismatch), and the call will still execute successfully because the contract only

validates that id exists and is QUEUED. In that case, a different user's request may be claimed/cancelled than the one the relayer intended.

So this is a wrong-request execution risk, not a receiver mutation risk.

A simple defense-in-depth fix is to add expected Receiver to claim and cancel, and revert on a mismatch.

Hope this clarifies your doubts.

AlturaTrade

https://github.com/AlturaTrade/contracts/commit/4d5fa8cd2b1c72ee8b6b94e846ff4cf7d812998b

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.