Adevar Labs

Prepared for:
Altura

December 02, 2025

Altura PreDeposit Contract Audit Report

Version: after-fix report

Table of Contents

1. Executive Summary

About Altura

This audit focuses on Altura's PreDeposit contract, an intermediate deposit mechanism deployed prior to the main Vault launch. The PreDeposit contract enables users to commit funds early by depositing assets and receiving preAVLT shares in return. Once the Vault is deployed and the claimStart timestamp is reached, users can redeem their preAVLT shares 1:1 for actual Vault shares after the PreDeposit owner transfers all collected funds to the Vault.

The main Altura Vault system is an ERC-4626 compliant NavVault where share pricing is determined by an external NavOracle based on performance of strategies.

Risk Profile

The following table summarizes the distribution of identified vulnerabilities by risk level:

Risk Level	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	1	1	0

Number of Enhancement Opportunities: 2

Overall Security Posture

Throughout the audit process, the Altura team demonstrated strong commitment to security and was highly responsive to findings. They actively engaged in discussions about identified issues and implemented fixes promptly, showing a proactive approach to ensuring the safety of user funds.

This audit follows a first audit of the Vault and Oracle contracts that is still in progress. It focuses exclusively on the PreDeposit contract, which allows users to deposit assets into an intermediate contract before the Vault is deployed.

After Fix Review

The key compromise risk has been resolved by implementing a two-step process with a 2-day delay before the vault address becomes effective, allowing users to review the configured contract and withdraw their assets if any risk is detected.

Before Fix Review

One Low severity issue was identified during this audit: user funds were under the complete control of the contract, presenting a risk to user funds in the event of admin key compromise.

Enhancement opportunities for gas optimization and input sanitization have been reported to the team.

Launch Recommendations

After Fix Review

The Low severity issue has been resolved.

Before Fix Review

To ensure a smooth and secure launch, we offer the following recommendations:

I. Strongly Recommended:

Fix the Low severity issue:

- Implement a two-step vault setup process allowing users to react in case of admin key compromise

II. Post-Launch Monitoring:

- Establish clear procedures for private key security management and post-incident response.

Audit Scope

- **Repository:** https://github.com/AlturaTrade/contracts/
- **Before fix review commit hash:** 2e3ea638f531ce2e7c6dcf551a9f92aeec50ff03
- **After fix review commit hash:** 049a5ca028ee9962c6c7af17fc210a0f519f9191
- **Files/Modules in Scope:**
  - PreDeposit.sol

2. Assumptions and Considerations

Audit Assumptions

I. The only supported vault asset will be USDT.

Centralization Risks

II. PreDeposit contract Owner role is a trusted and protocol-owned addresses.

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Note: Enhancements represent non-blocking improvements–typically usability, observability, or defense-in-depth tweaks that do not pose an immediate asset risk but would improve the product's reliability and/or user experience if implemented.

Impact

Impact reflects the potential consequences of the issue–particularly on **project funds**, **user funds**, and the **availability or integrity** of the protocol.

- **High Impact**: Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact**: Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact**: The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and exploit by an attacker, as well as how economically attractive the exploit is to an attacker.

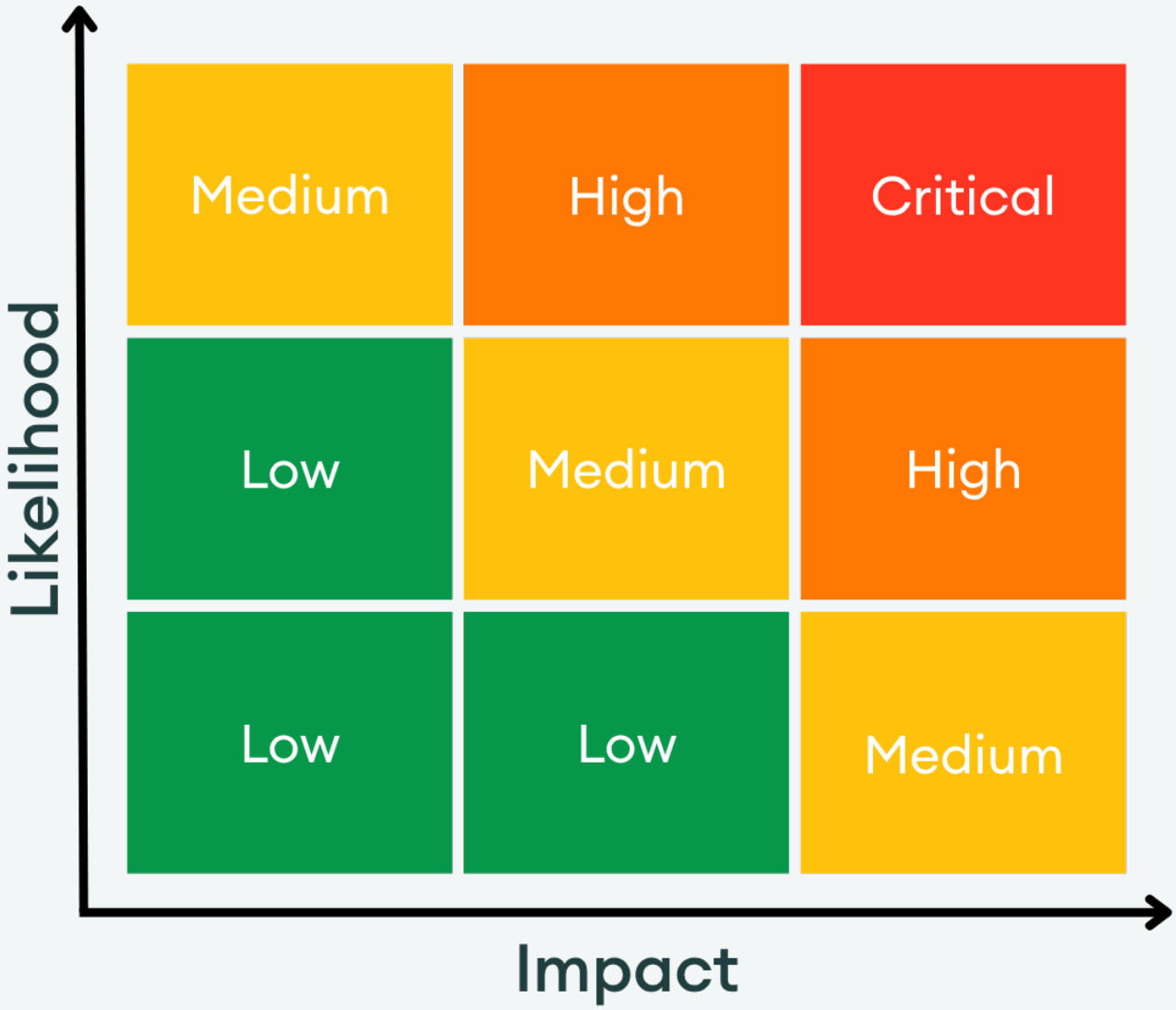
- **High Likelihood**: The vulnerability is trivially exploitable. This means it can be exploited by a wide range of actors without privileged access rights, with minimal capital requirements and low financial risks.
- **Medium Likelihood**: This type of vulnerability can be found and exploited with moderate effort. It might require a significant capital investment, but with manageable financial risk.
- **Low Likelihood**: Exploitation of these vulnerabilities is often technically unfeasible or requires highly specialized conditions. They may require extraordinary effort or a significant financial risk for an attacker, with a high chance of failure and minimal potential return.

Severity Classification Matrix

By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical**: High impact + high likelihood (e.g. a bug that could allow anyone to drain a substansial amount of protocol funds with minimal effort)
- **High**: High impact with medium likelihood, or medium impact with high likelihood
- **Medium**: Moderate impact and/or discoverability
- **Low**: Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

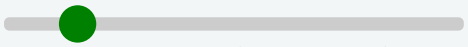
4. Findings

L01: Vault Address Update Lacks Timelock Protection Against Key Compromise

Status: Resolved

Impact:

Likelihood:



Severity: Low

Location: https://github.com/AdevarLabs/altura_code/tree/preDeposit-audit/contracts/PreDeposit.sol#L91-L96

>_ PreDeposit.sol SOLIDITY

```
89:     }
90:
91:     function setVault(address _vault) external onlyOwner {
92:         if (_vault == address(0)) revert BadAddress();
93:         if (address(vault) != address(0)) revert VaultAlreadySet();
94:         vault = IERC4626(_vault);
95:         emit VaultSet(_vault);
96:     }
97:
98:     function setClaimStart(uint256 ts) external onlyOwner {
```

Description:

The PreDeposit contract allows the owner to set the vault address through the setVault function with immediate effect and no timelock delay. Once users pre-deposit assets via preDeposit(), their funds are held in the contract until the claim period begins. The owner then calls depositAllToVault() to transfer all deposited assets to the configured vault address.

If the owner's private key is compromised, an attacker can call setVault() to point to a malicious vault contract they control where he would be able to transfer the users funds.

Recommendation:

Implement a delayed two-step process for the vault address update, and allow users to cancel their deposit during that timeframe.

Developer Response:

Fixed according to the recommendation.
Users now have a 2-day window where they can withdraw their assets before the new vault configuration becomes effective.

5. Enhancement Opportunities

E01: Unnecessary ownership transfer in PreDeposit constructor

Location: https://github.com/AdevarLabs/altura_code/blob/preDeposit-audit/contracts/PreDeposit.sol#L63-L63

>_ PreDeposit.sol — SOLIDITY

```
61:         PreAVLT _preToken = new PreAVLT();
62:         preToken = _preToken;
63:         _preToken.transferOwnership(address(this));
64:     }
65:
```

Description:

In the PreDeposit constructor, a new PreAVLT token contract is deployed with msg.sender as the initial owner which is the PreDeposit contract. Immediately after deployment, ownership is transferred from again to the PreDeposit contract itself.

Potential Benefit:

Save gas on deployment and make the code clearer.

Recommendation:

DIFF

```
constructor(address _asset, uint256 _claimStart)
    Ownable(msg.sender)
{
    if (_asset == address(0)) revert BadAddress();
    asset = IERC20(_asset);
    claimStart = _claimStart;
    PreAVLT _preToken = new PreAVLT();
    preToken = _preToken;
-    _preToken.transferOwnership(address(this));
}
```

Developer Response:

Fixed according to the recommendation.

E02: Validation of _claimStart will prevent admin errors

Location:
https://github.com/AdevarLabs/altura_code/blob/preDeposit-audit/contracts/PreDeposit.sol#L60

>_ PreDeposit.sol SOLIDITY

```
58:         if (_asset == address(0)) revert BadAddress();
59:         asset = IERC20(_asset);
60:         claimStart = _claimStart;
61:         PreAVLT _preToken = new PreAVLT();
62:         preToken = _preToken;
```

Description:

The _claimStart field in the constructor is not validated and is set to whatever value the contract is initialized with, making it possible to set it in the past.

Potential Benefit:

Prevent admin error.

Recommendation:

Ensure the _claimStart timestamp is in the future when setting it in the constructor. Additionally, a minimum claim period constant can be added and checked against the _claimStart.

Developer Response:

_claimStart can be updated through a newly added setClaimStart function.

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional enterprise and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by projects securing over $500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is specialized to provide thorough security assessments of Ethereum smart contracts. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your programs operate securely and as intended.

1. Program Context and Architecture Analysis

Our auditors begin by deeply examining your Ethereum smart contract's documentation and intended functionality. We meticulously map out contract interactions, transaction processing flows, and state management logic. Special attention is given to understanding how your program interfaces with critical EVM standards, such as ERC-20, ERC-721, or ERC-4626 tokens, and governance/timelock modules. Last but not least we check external integrations with other DeFi protocols and verify if the inputs and return values are handled properly.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for EVM's execution environment. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from Solana-specific issues, including but not limited to:

- Unauthorized tate variable modification
- Improper Access Control, role checks (onlyOwner, require(hasRole)) or msg.sender verification

- Misuse of low-level calls (call, delegatecall) or proxy/upgradeability patterns (e.g., storage collisions)
- Incorrect use of external calls, including reentrancy vectors
- Risks stemming from proxy initialization logic or denial-of-service related to gas limitations

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual security review of your Rust-based Solana programs. This involves a comprehensive line-by-line inspection of source code, focusing on common Solana vulnerabilities including, but not limited to:

- Missing or insufficient Access Control and Role Checks
- Inadequate msg.sender and transaction origin checks
- Incorrect handling of external calls and invocation privileges, particularly potential reentrancy risks
- Arithmetic and integer overflow or underflow errors
- Unsafe use of ABI Encoding/Decoding or low-level assembly
- Improper ERC-20/Token Handling (e.g., approval logic, flash loan vectors, token locking)
- Logic flaws in financial operations or state transitions
- Edge-case handling in function input validation
- Potential denial-of-service vectors related to gas usage and transaction execution

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions
- The fixes align closely with EVM best practices and secure coding guidelines

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the EVM development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, expressed or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.