LinumLabs

Flagship- Audit

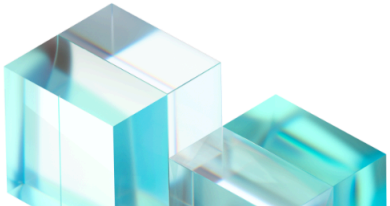
Prepared by Linum Labs AG

2025-05-30

linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Executive Summary

This document outlines the security review of the Flagship smart contracts. Linum Labs Auditing has identified several vulnerabilities that should be addressed before deploying smart contracts to any live chain. The report outlines these vulnerabilities and advises on how to fix them.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Protocol Summary

Overview

Flagship is an innovative platform creating AI-managed crypto portfolios, aiming for mass adoption with its mobile-first approach. At its core is a "Crypto Brain" that leverages autonomous AI managers, adaptive learning, and real-time data to provide 24/7 crypto portfolio strategizing.

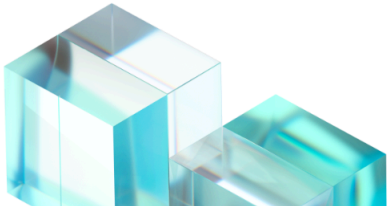
Audit Scope

./src/FYIToken.sol
./src/Presale.sol

Audit Results

Summary

Repository	https://github.com/LinumLabs/flagship-contracts
Commit	b8dcf54b1956590bc7933b555407b1318584df03
Timeline	May 28th - May 30th



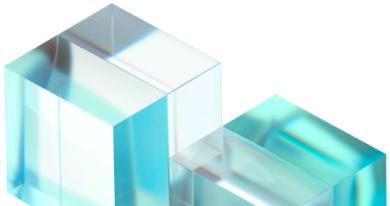
linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Issues Found

Bug Severity	Count
Critical	0
High	0
Medium	3
Low	2
Informational	4
Total Findings	9

Summary of Issues

Description	Severity	Status
Lack of Comprehensive Chainlink Stale Price Checks.	Medium	Resolved
Potential Overselling of Tokens Due to Incorrect Supply Check.	Medium	Resolved
Incomplete Pausability Implementation for Token Operations.	Medium	Resolved
Inefficient Tier Progression for Large Purchases.	Low	Resolved
Unpriced Token Purchases if No Tiers Initialized.	Low	Resolved



Redundant Initialization of claimable State Variable.	Informational	Resolved
Unutilized Event Declarations.	Informational	Resolved
Single-Step Ownership Transfer.	Informational	Resolved
Redundant Check for totalTokensSold in setSaleToken.	Informational	Resolved

Issues

Medium Severity

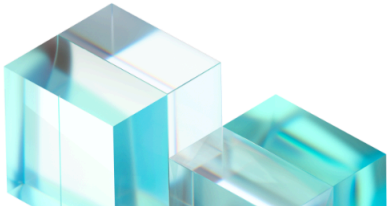
1. Lack of Comprehensive Chainlink Stale Price Checks.

Description: The `getStablecoinPrice()` function retrieves the latest price from the Chainlink `AggregatorV3Interface` but only checks `require(price > 0, "Invalid price feed");`. It lacks crucial checks for stale data, specifically verifying the `updatedAt` timestamp against `block.timestamp` and ensuring `answeredInRound` is not less than `roundId`. While `whitelistStablecoin` performs these checks, they are missing in the `getStablecoinPrice` function which is called during `purchaseTokens`.

Potential Risk: Transactions could proceed using outdated or manipulated price data, leading to incorrect calculations for `paymentAmount` and potential financial losses for users or the contract.

Suggested Mitigation: Add `updatedAt` and `answeredInRound` checks within the `getStablecoinPrice()` function (similar to those in `whitelistStablecoin`) to ensure the retrieved price is fresh and valid before being used for calculations.

Flagship: Fixed in PR.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Linum Labs: Verified.

2. Potential Overselling of Tokens Due to Incorrect Supply Check.

Description: The `purchaseTokens` function uses `saleToken.totalSupply()` to determine the maximum available tokens for sale: `require(totalTokensSold + amount <= saleToken.totalSupply(), "Not enough tokens available");`. However, according to the `FYIToken.sol` contract, there are specific allocations (e.g., `Allocation.SALES`) that define the actual amount of tokens designated for presales, not the total supply of the token. This means the `Presale` contract could potentially sell more tokens than are actually allocated for the sale.

Potential Risk: The contract could oversell tokens, leading to a shortage when users attempt to claim. This results in an inability to fulfill commitments, damaging reputation and potentially requiring manual intervention to resolve.

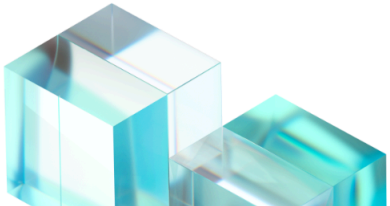
Suggested Mitigation: The `Presale` contract should directly query the `allocations[Allocation.SALES]` getter on the `FYIToken` contract instance. This retrieved allocation amount should then be used as the maximum limit for `totalTokensSold` instead of `saleToken.totalSupply()`.

Flagship: Fixed in PR.

Linum Labs: Verified.

3. Incomplete Pausability Implementation for Token Operations.

Description: The `FYIToken.sol` contract inherits `Pausable` and includes `pause()` and `unpause()` functions. However, the `whenNotPaused` modifier is only applied to the `mint()` function. This means that while new tokens cannot be minted when paused, core ERC20 functionalities like `transfer()`, `transferFrom()`, `approve()`, and `ERC20Burnable` functionalities such as `burn()` and `burnFrom()` remain active and usable even when the contract is in a paused state.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Potential Risk: The contract cannot truly "pause" all token activities, leading to a false sense of security during emergencies or upgrades. Critical token operations might proceed unexpectedly during a supposed pause.

Suggested Mitigation: Apply the `whenNotPaused` modifier to the relevant inherited ERC20 functions (e.g., `_transfer`, `_approve`) and `ERC20Burnable` functions (e.g., `_burn`) by overriding them. This ensures that all critical token functionalities are paused when the contract's `_paused` state is active, providing comprehensive pausability.

Flagship: Fixed in PR.

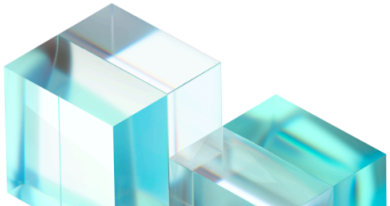
Linum Labs: Verified.

Low Severity

4. Inefficient Tier Progression for Large Purchases.

Description: The `getUserCurrentTier()` function relies on `userContributions[msg.sender]` to determine the user's current tier. In the `purchaseTokens()` function, `userContributions[msg.sender]` is updated *after* the `getUserCurrentTier()` call, meaning the current transaction's contribution is not factored into the tier calculation for the *current* purchase. This leads to scenarios where a user making a bulk purchase might not immediately qualify for the better price associated with a higher tier, even if their total contribution *after* the current purchase would place them in that tier. They would effectively pay the price of their *previous* tier for the current bulk purchase.

Potential Risk: Users might miss out on better pricing for their current bulk purchase, leading to dissatisfaction or a perception of unfairness.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Suggested Mitigation: Implement a progressive pricing mechanism within the purchaseTokens function. Instead of simply getting a single currentTier and pricePerToken for the entire purchase, calculate the cost by iterating through the tiers. For a given purchase, determine how much of the purchase falls into the user's current tier, then if the purchase crosses into a new tier, calculate the cost for the amount in the new tier, and so on. This ensures that the price paid accurately reflects the tiered structure for the entire contribution, even within a single transaction.

Flagship: Fixed in PR.

Linum Labs: Verified.

5. Unpriced Token Purchases if No Tiers Initialized.

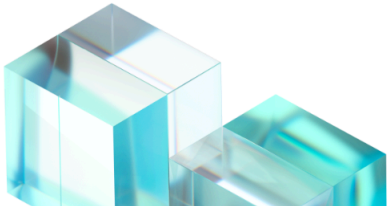
Description: The `Presale` contract allows for the sale of tokens through a tiered pricing system. The `getUserCurrentTier()` function determines a user's applicable tier, and `getTierPrice()` retrieves the price for that tier. However, if the contract owner fails to initialize any tiers using the `addTier()` or `updateTier()` functions (meaning `tierCount` remains `0`), a critical vulnerability arises.

Potential Risk: If no pricing tiers are set, the contract calculates a token price of zero. This allows users to purchase tokens for free.

Suggested Mitigation: Add a **`require` statement** to explicitly check that the calculated `pricePerToken` is greater than zero.

Flagship: Fixed in PR.

Linum Labs: Verified.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Informational Severity

6. Redundant Initialization of claimable State Variable.

Description: The `claimable` state variable is of type `bool` and is explicitly initialized to `false` within the `initialize()` function. In Solidity, boolean state variables are automatically initialized to their default value of `false` if no explicit assignment is made.

Potential Risk: While not a direct security vulnerability, redundant initializations can clutter the codebase, slightly increase bytecode size, and potentially cause confusion for developers reviewing the contract. It adds an unnecessary operation during deployment without altering the contract's intended behavior.

Suggested Mitigation: Remove the line `claimable = false;` from the `initialize()` function. The `claimable` variable will still default to `false`, achieving the same intended initial state without explicit assignment.

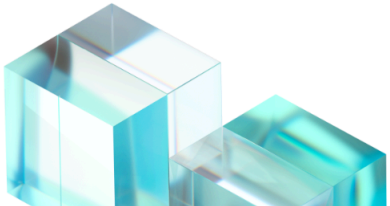
Flagship: Fixed in PR.

Linum Labs: Verified.

7. Unutilized Event Declarations.

Description: The events `PresaleFinalized`, `PriceFeedUpdated`, and `MaxDeviationUpdated` are declared in the contract but are never emitted by any function. Events are crucial for off-chain monitoring, debugging, and understanding contract state changes. Declaring them without emitting them makes them functionally useless.

Potential Risk: Lack of transparency for off-chain monitoring; DApps cannot track these specific state changes. Minor code bloat.



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

Suggested Mitigation: Either emit these events at appropriate points or remove their declarations if they are not intended to be used.

Flagship: Fixed in PR.

Linum Labs: Verified.

8. Single-Step Ownership Transfer.

Description: Both `FYIToken.sol` and `Presale.sol` contracts implement a single-step ownership transfer mechanism. `FYIToken.sol` utilizes the standard `Ownable` contract, where ownership is immediately transferred upon execution of `transferOwnership()`. `Presale.sol` is an upgradeable contract vand uses `OwnableUpgradeable`. Similar to `Ownable`, `OwnableUpgradeable` also employs a single-step transfer, meaning ownership changes hands immediately without requiring confirmation from the new owner.

Potential Risk: A single-step ownership transfer poses a significant risk across both contracts. An accidental error, such as a typo in the new owner's address, can lead to the permanent loss of control over the respective contract.

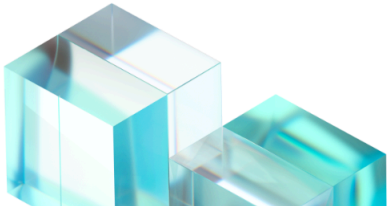
Suggested Mitigation: To enhance security and prevent accidental loss of control for both contracts, it is strongly recommended to implement a two-step ownership transfer process.

Flagship: Fixed in PR.

Linum Labs: Verified.

9. Redundant Check for totalTokensSold in setSaleToken.

Description: The `setSaleToken` function includes two `require` statements to prevent changing the sale token after the presale has started: `require(totalTokensSold == 0, "Cannot change token after sales have started");`



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

```
require(block.timestamp < presaleStartTime, "Cannot change token after presale starts");
```

The second check, `block.timestamp < presaleStartTime`, implicitly covers the first. If the current timestamp is before the presale start time, it guarantees that no tokens could have been sold (`totalTokensSold` would necessarily be `0`).

Potential Risk: While not a security vulnerability, redundant checks can slightly increase gas costs and bytecode size, making the code marginally less efficient and potentially harder to read or maintain due to unnecessary complexity.

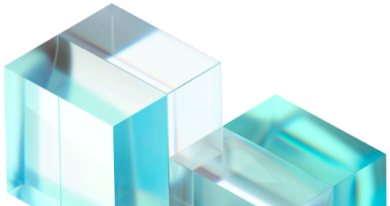
Suggested Mitigation: Remove the `require(totalTokensSold == 0, "Cannot change token after sales have started");` line. The check `require(block.timestamp < presaleStartTime, "Cannot change token after presale starts");` is sufficient to ensure the sale token cannot be changed once the presale period has begun, which inherently means `totalTokensSold` will still be zero.

Flagship: Fixed in PR.

Linum Labs: Verified.

Disclaimer

This report is based on the materials and documentation provided to Linum Labs Auditing for the purpose of conducting a security review, as outlined in the Executive Summary and Files in Scope sections. It's important to note that the results presented in this report may not cover all vulnerabilities. Linum Labs Auditing provides this review and report on an as-is, where-is, and as-available basis. By accessing and/or using this report, along with associated services, products,



linumlabs.com
info@linumlabs.com
+1 (650) 534-2011

protocols, platforms, content, and materials, you agree to do so at your own risk. Linum Labs Auditing disclaims any liability associated with this report, its content, and any related services and products, to the fullest extent permitted by law. This includes implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Linum Labs Auditing does not warrant, endorse, guarantee, or assume responsibility for any third-party products or services advertised or offered through this report, its content, or related services and products. Users should exercise caution and use their best judgment when engaging with third-party providers. It's important to clarify that this report, its content, access, and/or usage thereof, including associated services or materials, should not be considered or relied upon as financial, investment, tax, legal, regulatory, or any other form of advice.