GA GUARDIAN

Peapods

Leveraged Volatility Farming

Security Assessment

November 11th, 2024

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Nicholas Chew, Mark Jonathas, Osman Ozdemir, Zdravko Hristov, Michael Lett

Client Firm Peapods

Final Report Date November 11, 2024

Audit Summary

Peapods engaged Guardian to review the security of its leveraged volatility farming updates. From the 9th of September to the 3rd of October, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 25 High/Critical issues were uncovered and promptly remediated by the Peapods team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the primary functionality described for the LVF system.

Security Recommendation Given the number of High and Critical issues detected, Guardian supports a secondary security review of the protocol at a finalized frozen commit. Furthermore, the Peapods team should increase units tests across the codebase, as well as integration tests between the LVF and FraxLend systems. The engagement exposed multiple blind spots within the auto-compounding logic and the FraxLend interaction that should be thoroughly tested, and the Foundry testing infrastructure that was built during the audit can be utilized.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Arbitrum, Ethereum**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/peapods-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Peapods
Language	Solidity
Codebase	https://github.com/peapodsfinance/contracts, https://github.com/peapodsfinance/fraxlend
Commit(s)	Core / FraxLend Initial Commit: cebeb47 / dc76f7b Core / FraxLend Final Commit: f1a3182 / 4ea61dc

Audit Summary

Delivery Date	November 11, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	11	0	0	0	2	9
High	14	0	0	2	0	12
Medium	45	0	0	11	2	32
Low	31	0	0	15	2	14

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Peapods, fuzz-testing with Echidna was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
POD-1	LeverageManager::_acquireBorrowTokenForRepayment should never Uniswap revert	✅	❌	❌	10M+
POD-2	LendingAssetVault::deposit/mint share balance of receiver should increase	✅	❌	❌	10M+
POD-3	LendingAssetVault::withdraw/redeem share balance of user should decrease	✅	✅	✅	10M+
POD-4	vaultUtilization[_vault] == FraxLend.convertToAssets(LAV shares) post-update	✅	❌	❌	10M+
POD-5	LendingAssetVault::totalAssetsUtilized totalAssetsUtilized == sum(all vault utilizations)	✅	✅	✅	10M+
POD-6	LendingAssetVault::whitelistDeposit totalAvailableAssets() should increase	✅	✅	✅	10M+
POD-7	LendingAssetVault::whitelistDeposit vault utilization should decrease accurately	✅	❌	❌	10M+
POD-8	LendingAssetVault::whitelistDeposit total utilization should decrease accurately	✅	❌	❌	10M+
POD-9	LendingAssetVault::whitelistWithdraw totalAvailableAssets() should decrease	✅	✅	✅	10M+
POD-10	LendingAssetVault::whitelistWithdraw vault utilization should increase accurately	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POD-11	LendingAssetVault::whitelistWithdraw total utilization should increase accurately	✅	✅	✅	10M+
POD-12	LendingAssetVault::global total assets == sum(deposits + donations + interest accrued - withdrawals)	✅	❌	❌	10M+
POD-13	LendingAssetVault::withdraw/redeem User can't withdraw more than their share of total assets	✅	❌	✅	10M+
POD-14a	LendingAssetVault::donate Post-donation shares shouldn't have increased, but totalAssets should have by donated amount	✅	❌	✅	10M+
POD-14b	LendingAssetVault::donate Post-donation shares shouldn't have increased, but totalAssets should have by donated amount	✅	✅	✅	10M+
POD-15	LendingAssetVault::global FraxLend vault should never more assets lent to it from the LAV that the allotted _vaultMaxPerc	✅	✅	✅	10M+
POD-16	LendingAssetVault::whitelistDeposit Post-state utilization rate in FraxLend should have decreased (called by repayAsset in FraxLend) (utilization rate retrieved from currentRateInfo public var)	✅	✅	✅	10M+
POD-17	LendingAssetVault::whitelistWithdraw Post-state utilization rate in FraxLend should have increased or not changed (if called within from a redeem no change, increase if called from borrowAsset)	✅	✅	✅	10M+
POD-18a	LeverageManager::addLeverage Post adding leverage, there totalBorrow amount and shares, as well as utilization should increase in Fraxlend	✅	✅	✅	10M+
POD-18b	LeverageManager::addLeverage Post adding leverage, there totalBorrow amount and shares, as well as utilization should increase in Fraxlend	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POD-19a	LeverageManager::removeLeverage Post removing leverage, there totalBorrow amount and shares, as well as utilization should decrease in Fraxlend	✅	✅	✅	10M+
POD-19b	LeverageManager::removeLeverage Post removing leverage, there totalBorrow amount and shares, as well as utilization should decrease in Fraxlend	✅	✅	✅	10M+
POD-20	Post adding leverage, there should be a higher supply of spTKNs (StakingPoolToken)	✅	✅	✅	10M+
POD-21	Post adding leverage, there should be a higher supply of aspTKNs (AutoCompoundingPodLp)	✅	✅	✅	10M+
POD-22	Post adding leverage, the custodian for the position should have a higher userCollateralBalance	✅	✅	✅	10M+
POD-23	Post removing leverage, there should be a lower supply of spTKNs (StakingPoolToken)	✅	❌	❌	10M+
POD-24	Post removing leverage, there should be a lower supply of aspTKNs (AutoCompoundingPodLp)	✅	✅	✅	10M+
POD-25	Post removing leverage, the custodian for the position should have a lower userCollateralBalance	✅	✅	✅	10M+
POD-26	FraxLend: cbr change with one large update == cbr change with multiple, smaller updates	✅	❌	❌	10M+
POD-27	LeverageManager contract should never hold any token balances	✅	✅	✅	10M+
POD-28	FraxlendPair.totalAsset should be greater or equal to vaultUtilization (LendingAssetVault)	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POD-29	LendingAssetVault::global totalAssets must be greater than totalAssetUtilized”	✅	✅	✅	10M+
POD-30	repayAsset should not lead to to insolvency	✅	✅	✅	10M+
POD-31	staking pool balance should equal token reward shares	✅	✅	✅	10M+
POD-32	FraxLend: (totalBorrow.amount) / totalAsset.totalAmount(address(externalAsset Vault)) should never be more than 100%	✅	✅	✅	10M+
POD-33	FraxLend: totalAsset.totalAmount(address(0)) == 0 -> totalBorrow.amount == 0	✅	✅	✅	10M+
POD-34	AutoCompoundingPodLP: mint() should increase asp supply by exactly that amount of shares	✅	✅	✅	10M+
POD-35	AutoCompoundingPodLP: deposit() should decrease user balance of sp tokens by exact amount of assets passed	✅	✅	✅	10M+
POD-36	AutoCompoundingPodLP: redeem() should decrease asp supply by exactly that amount of shares	✅	✅	✅	10M+
POD-37	AutoCompoundingPodLP: withdraw() should increase user balance of sp tokens by exact amount of assets passed	✅	✅	❌	10M+
POD-38	AutoCompoundingPodLP: mint/deposit/redeem/withdraw() spToken total supply should never decrease	✅	✅	✅	10M+
POD-39	AutoCompounding should not revert with Insufficient Amount	✅	❌	❌	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POD-40	AutoCompounding should not revert with Insufficient Liquidity	✅	❌	❌	10M+
POD-41	AutoCompoundingPodLP: redeem/withdraw() should never get an InsufficientBalance or underflow/overflow revert	✅	✅	✅	10M+
POD-42	custodian position is solvent after adding leverage and removing leverage	✅	✅	✅	10M+
POD-43	TokenReward: global: getUnpaid() <= balanceOf reward token	✅	✅	✅	10M+
POD-44	LVF: global there should not be any remaining allowances after each function call	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Incorrect Price From aspTKN Oracle	Logic Error	Critical	Resolved
C-02	Compounding Of Rewards To LP Failure	Logic Error	Critical	Partially Resolved
C-03	spTKN Oracle Can Be Manipulated With Donation	Oracle Manipulation	Critical	Resolved
C-04	removeLeverage DoS Due To Inconsistent Amounts	DoS	Critical	Resolved
C-05	Accounting Error In totalAvailableAssetsForVault	Logic Error	Critical	Resolved
C-06	User Voting Shares Can Be Burned By Others	Logic Error	Critical	Resolved
C-07	Inflation Attack In LendingAssetVault	Logic Error	Critical	Resolved
C-08	aspTKNOracle Can Be Manipulated With Donation	Oracle Manipulation	Critical	Partially Resolved
C-09	Oracle Precision Error Due To Token Decimals	Arithmetic Error	Critical	Resolved
C-10	UniswapDexAdapter.swapV2Single Doesn't Work	DoS	Critical	Resolved
C-11	Incorrect addInterest Interface	Integration	Critical	Resolved
C-12	pTKN Share Siphoning Via FlashMint	Logic Error	High	Resolved
H-01	Lack Of Access Control In redeemFromVault	Access control	High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
H-02	Whitelist Actions Should Update All Vaults	Logic Error	High	Resolved
H-03	Staking Pool Rewards Sniping Is Possible	Logic Error	High	Acknowledged
H-04	Loss Of Rewards Due Two Step Swap Failure	Logic Error	High	Resolved
H-05	Rewards Are Lost For aspToken	Logic Error	High	Resolved
H-06	Malicious Function Input In Add/Remove Leverage	Logic Error	High	Resolved
H-07	Reward Sniping With AutoCompounder	Logical Error	High	Acknowledged
H-08	removeLeverage Inaccurate Share Calculation	Logic Error	High	Resolved
H-09	Liquidators Can Avoid Bad Debt Socialization	Logic Error	High	Resolved
H-10	DoS In _withdrawToVault Due To Underflow	DoS	High	Resolved
H-11	Assets Can Be Borrowed/Repaid While Paused	Logic Error	High	Resolved
H-12	Lack Of _selfLendingPairPod Validation	Validation	High	Resolved
H-13	Flash Mint Manipulates Supply	Logical Error	High	Resolved
M-01	Insufficient Token Amounts Lead To Swap Error	Logic Error	Medium	Partially Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-02	Underflow In _updateAssetMetadataFromVault	Arithmetic Error	Medium	Partially Resolved
M-03	Precision Loss Leads To Reverts In TokenRewards	Precision	Medium	Resolved
M-04	Positions Are Lost When lendingPair Is Changed	Logical Error	Medium	Resolved
M-05	Flash Loan Repayments Fail During addLeverage	Logical Error	Medium	Resolved
M-06	Borrowers Pay For Paused Interest	Logical Error	Medium	Resolved
M-07	Improper Slippage And Deadline During Deposit	Logic Error	Medium	Acknowledged
M-08	AutoCompoundingPodLp Is Not EIP Compliant	ERC4626	Medium	Resolved
M-09	Rewards Not Updated Prior To Fee Change	Logic Error	Medium	Resolved
M-10	Vault Whitelist Can Be Set One Above Max	Logic Error	Medium	Resolved
M-11	Inflated Admin Fee	Logic Error	Medium	Resolved
M-12	Same Heartbeat For Multiple Oracles	Oracles	Medium	Resolved
M-13	No Circuit Breaker Checks In ChainlinkOracle	Oracles	Medium	Resolved
M-14	DOS Of Borrow & Redeem	Logic Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-15	Feeds With > 18 Decimals Are Problematic	Math	Medium	Resolved
M-16	addLiquidity Fails For Fee-On-Transfer Tokens	Logic Error	Medium	Acknowledged
M-17	Single Token Pod Assumption	Logic Error	Medium	Acknowledged
M-18	Wrong Price Calculations If T0 Is baseToken	Protocol	Medium	Resolved
M-19	Improper Deadline For Fraxlend Swaps	Logic Error	Medium	Resolved
M-20	FraxVault Incompatible With Non-Standard Tokens	Logical Error	Medium	Acknowledged
M-21	removeLeverage Could Fail For Self-Lending Pairs	Logical Error	Medium	Resolved
M-22	USDC Blacklist Prevents Transfers	Logic Error	Medium	Resolved
M-23	Uniswap Ticks Rounding	Math	Medium	Resolved
M-24	Arbitrage From Deviation In Oracle Price	Oracles	Medium	Acknowledged
M-25	mint In Fraxlend Rounds In Users' Favour	Rounding	Medium	Resolved
M-26	Oracle Incompatible With Non-Standard Tokens	Underflow	Medium	Resolved
M-27	Fee-on-transfer Tokens Are Not Supported	Logic Error	Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-28	Missing Check For Sequencer Downtime	Oracle Manipulation	Medium	Resolved
M-29	ASP Insufficient Liquidity DOS	DoS	Medium	Resolved
M-30	User's Lockup Period Should Not Change Midway	Logic Error	Medium	Resolved
M-31	Asp Rewards Can Be Sandwiched	Logic Error	Medium	Resolved
M-32	Incorrect Autocompounding Asset And Shares Conversions	Logic Error	Medium	Resolved
M-33	LendingAssetVault Asset/Share Conversion Error	Logic Error	Medium	Resolved
M-34	VotingPool Pods Priced Equally	Protocol	Medium	Acknowledged
M-35	VotingPool Incompatible With Non-Standard Tokens	Integration	Medium	Acknowledged
M-36	redeemFromVault DOS	DoS	Medium	Resolved
M-37	Vaults' Utilization Not Updated After Bad Debt	Logic Error	Medium	Resolved
M-38	Same TWAP For Multiple V3 Pools	Oracles	Medium	Acknowledged
M-39	_getPairedTknAmt 0 Bond Slippage	Logic Error	Medium	Acknowledged
M-40	getPairAccounting Includes LAV Assets	Logic Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-41	Donation Increases Share Supply	Logical Error	Medium	Resolved
M-42	Not Updating Pairs Will Break LAV	Protocol	Medium	Resolved
M-43	Leverage Doesn't Work When FOT Is Enabled	DoS	Medium	Acknowledged
M-44	Swap Error Handling Causes DOS Of AutoCompounder	Logic Error	Medium	Resolved
M-45	DOS Of depositFromPairedLpToken	Logic Error	Medium	Resolved
L-01	Wrong Address Assignment	Logic Error	Low	Resolved
L-02	Oracle Incompatible With Existing Pod Contracts	Integration	Low	Acknowledged
L-03	POD Ratio Can Be Manipulated	Protocol	Low	Resolved
L-04	PodFlashSource Incompatible With Existing Pods	Integration	Low	Acknowledged
L-05	Malicious Pod Could Allow Reentrancy	Reentrancy	Low	Acknowledged
L-06	LendingAssetVault Is Not EIP-4626 Compliant	ERC4626	Low	Resolved
L-07	Users Avoid Debonding Fee	Logic Error	Low	Partially Resolved
L-08	_clBaseFeed Should Be Set When BASE != USD	Oracles	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	Dormant Token Rewards	Logic Error	Low	Resolved
L-10	Unused Code In LAV	Best Practices	Low	Resolved
L-11	Lenders Can Avoid Socialization Of Bad Debt	Logical Error	Low	Acknowledged
L-12	Removing Pairs In LAV Leaves Dirty State	Logic Error	Low	Resolved
L-13	Whitelist Update Does Not Update All Vaults	Logic Error	Low	Acknowledged
L-14	Approved Parties Cannot removeLeverage	Protocol	Low	Resolved
L-15	Stale Price Causes Division By 0	Logic Error	Low	Acknowledged
L-16	Oracle Reverts On Stale Price	Oracles	Low	Acknowledged
L-17	Pod Flashloans Prevented Via Lock	Logic Error	Low	Acknowledged
L-18	VotingPool Unstake Rounding	Math	Low	Acknowledged
L-19	Possible Inflation Attack In AutoCompounder	Logic Error	Low	Partially Resolved
L-20	Approved Address Gets The Flash Loan Refund	Logic Error	Low	Resolved
L-21	getFullUtilizationInterest() Calculation	Logic Error	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-22	Unused Cached Value	Logic Error	Low	Resolved
L-23	bondWeightedFromNative Fails For Multi-Asset Pod	Logic Error	Low	Acknowledged
L-24	Both buy And sell Fees Can Be Applied	Logical Error	Low	Resolved
L-25	DOS Of Native Bond And Stake	Logic Error	Low	Resolved
L-26	Unsuccessful Asp Reward Swaps	Protocol	Low	Acknowledged
L-27	Asp Rewards Impact removeLeverage	Protocol	Low	Resolved
L-28	LVF Bond Rounding	Protocol	Low	Acknowledged
L-29	Typos	Logic Error	Low	Resolved
L-30	Linear Interest Rate Will DOS Fraxlend	DoS	Low	Acknowledged
L-31	overrideBorrowAmt May Be Used Maliciously	Logical Error	Low	Acknowledged

C-01 | Incorrect Price From aspTKN Oracle

Category	Severity	Location	Status
Logic Error	● Critical	aspTKNMinimalOracle.sol: 51	Resolved

Description PoC

The `getPrice` function should return price as `(aspTKN / pairedLPToken)` which is consumed by the Fraxlend `isSolvent` function to determine a borrower's LTV.

However, the calculation is incorrect because it takes price from the `spTKN` oracle and divides it by `_aspTknPerSpTkn` when it should be multiplying instead. Therefore, the price returned is always incorrect and borrower's LTV is miscalculated in Fraxlend.

Recommendation

Instead of `_priceLow = (_priceLow * _assetFactor) / _aspTknPerSpTkn;` do `_priceLow = (_priceLow * _aspTknPerSpTkn) / _assetFactor;`

Resolution

Peapods Team: Resolved in the following code change.

C-02 | Compounding Of Rewards To LP Failure

Category	Severity	Location	Status
Logic Error	● Critical	AutoCompoundPodLp.sol	Partially Resolved

Description PoC

Function _processRewardsToPodLp is called on every major flow such as deposit and withdrawal to compound any earned rewards to the LP token and back into the AutoCompoundingPodLp. The amounts passed to indexUtils.addLPAndStake will be the entire balance of POD in the AutoCompoundingPodLp, and half of the paired lp token that's obtained from the reward tokens with a V3 swap.

The issue is that the token A and token B amounts can be wildly different from their current reserve ratio in the pool, causing the desired token inputs to fail the calculated amountAMin and amountBMin passed to Uniswap V2.

There can be a multitude of reasons why few paired LP tokens are to be added, such as small reward distribution since the last reward claim and/or V3 swap manipulation in swapV3Single since 0 slippage is passed. An attacker is not necessary for the UniV2 revert to occur.

Attackers can also inflate the balances with token donations to trigger this revert as well. This DoS will occur even if LP_SLIPPAGE was drastically increased. Ultimately, all core functionalities of the AutoCompoundingPodLp can be prevented, and users can lose assets due to the inability to withdraw.

Recommendation

In _pairedLpTokenToPodLp, consider performing some sanity checks to ensure tokens are in the correct ratio before calling IndexUtils.addLPAndStake. Furthermore, considering wrapping the auto-compounding in a try-catch.

Resolution

Peapods Team: Resolved in the following code change.

Guardian Team: The implemented single-sided LP formula is incorrectly implemented, using _fullAmt instead of _r where necessary. This will make the result of _pairedSwapAmt larger than _amountIn, causing an underflow and blocking all rewards processing.

C-03 | spTKN Oracle Can Be Manipulated With Donation

Category	Severity	Location	Status
Oracle Manipulation	● Critical	spTKNMinimalOracle.sol: 141-145, 266-271	Resolved

Description PoC

The price of a spTKN is affected by the amount and price of the underlying token in the pod. This is accounted for in _accountForCBRInPrice which does:

(_amtUnderlying * IERC20(_underlying).balanceOf(_pod) * 10 * IERC20Metadata(_pod).decimals()) / IERC20(_pod).totalSupply() / 10 * IERC20Metadata(_underlying).decimals();

The problem lies in using the balance of underlying the pod, which can be easily manipulated through a donation of the underlying token to the pod. This would increase the value of spTKN and subsequently the aspTKN, which is the collateral token in FraxPairLend.

Although the attacker loses the donated tokens, they can manipulate the oracle pricing to borrow more tokens from the lending protocol, exploiting the system. Low liquidity pods are more susceptible to such attacks.

Recommendation

Instead of using balanceOf, use a storage variable to keep track of the balance of underlying tokens in a pod.

Resolution

Peapods Team: Resolved in the following code change.

C-04 | removeLeverage DoS Due To Inconsistent Amounts

Category	Severity	Location	Status
DoS	● Critical	LeverageManager.sol: 169-171	Resolved

Description

When removing leverage, the user provides the `_borrowAssetAmt` to be flash loaned. This amount is then used to calculate `_borrowSharesToRepay`, with the calculation performed by rounding up.

Additionally, the `LeverageManager` grants approval to the Fraxlend pair for exactly the `_borrowAssetAmt`. Then, on the Fraxlend side, amount to repay is recalculated using this `_borrowSharesToRepay`.

However, this calculation also rounds up. `_amountToRepay = _totalBorrow.toAmount(_shares, true);`. Because of rounding up twice during this transaction flow, the `_amountToRepay` ends up being higher than the flash-loaned `_borrowAssetAmt`.

When the internal `_repayAsset` function attempts to transfer `_amountToRepay` from `LeverageManager` to Fraxlend pair, it fails because the `LeverageManager` neither holds that amount of the borrow asset nor has given that amount of approval to the Fraxlend pair.

Recommendation

Consider passing in the shares in `removeLeverage` and calculate the flash loan amount from the shares to mimic FraxLend logic.

Resolution

Peapods Team: Resolved in the following code change.

C-05 | Accounting Error In totalAvailableAssetsForVault

Category	Severity	Location	Status
Logic Error	● Critical	LendingAssetVault.sol: 85	Resolved

Description PoC

totalAvailableAssetsForVault should return the available assets that a FraxlendPair vault can pull from LendingAsssetVault.However, several accounting errors exist resulting in the under-calculation of available assets. Consider these two examples of a single whitelisted vault with 100% allocation:

LendingAssetVault has 10 DAI of which 6 DAI has been withdrawn into FraxPair Vault

Example 1

• totalAvailableAssetsForVault will return 0 when it should return 4 instead

Example 2

• LendingAssetVault has 10 DAI of which 4 DAI has been withdrawn into FraxPair Vault
• totalAvailableAssetsForVault will return (10 - 4) - 4 = 2 when it should return 6 instead

As FraxLendPair relies heavily on this function to obtain available assets from LendingAssetVault this results in: 1) preventing further whitelistWithdraw after 50% of assets are withdrawn, 2) inflating utilization rate in FraxlendPair and increase interest charged to borrowers, 3) deposits allowed above the depositLimit.

Recommendation

Update the function to:

```
uint256 _overallAvailable = totalAvailableAssets();
uint256 _vaultMax = ((_totalAssets * _vaultMaxPerc[_vault]) / PERCENTAGE_PRECISION);
uint256 _totalVaultAvailable = _vaultMax > vaultUtilization[_vault] ?
      _vaultMax - vaultUtilization[_vault] : 0;
_totalVaultAvailable = _overallAvailable < _totalVaultAvailable ?
      _overallAvailable : _totalVaultAvailable;
return _totalVaultAvailable;
```

Resolution

Peapods Team: Resolved in the following code change.

C-06 | User Voting Shares Can Be Burned By Others

Category	Severity	Location	Status
Logic Error	● Critical	VotingPool.sol	Resolved

Description

The update function can be called by anyone to update another user's stake position. The issue lies in `_update` which burns a user's share balance if the CBR had decreased from the time when the user first staked.

Consider this example:

- Alice stakes 10 pTKNs and received 10 voting shares. CBR is 1.0
- CBR drops to 0.8 due to external factors
- Bob calls update on Alice's position. 2 shares are burned from her

If Bob was unable to call update on Alice's position, she could choose to do nothing and preserve her shares, potentially waiting for CBR to recover before performing more staking actions.

Furthermore, the CBR can be be drastically decreased with a flashloan from the pod, which would allow Bob to initially stake, flashloan to decrease CBR, and update Alice's position to burn all of her voting power.

Consequently, Bob can have all the voting power and claim all the rewards at the expense of other stakers.

Recommendation

Do not allow update to be called on another user's position and consider limiting direct balance checks. Also, do re-consider the design of the burn during _update as it will deter users from staking if their previous shares are burnt due to a change in CBR.

Resolution

Peapods Team: Resolved in the following code change.

C-07 | Inflation Attack In LendingAssetVault

Category	Severity	Location	Status
Logic Error	● Critical	LendingAssetVault.sol	Resolved

Description PoC

The classic inflation attack during the first deposit is possible in the `LendingAssetVault` (LAV) through the `donate` function.

Attack scenario:
• LAV is created. attacker deposits 1 wei of assets and receives 1 share
• Attacker observes User depositing 100e18 of assets and frontruns with a donation of 100e18 assets
• User deposits 100e18 but receives 0 shares due to rounding down
• Attacker redeems 1 share and receives all assets in the vault (200e18 + 1 wei)
• User loses all deposits

Recommendation

Consider removing the donate function. Or else, consider other forms of protection against inflation attack, see https://blog.openzeppelin.com/a-novel-defense-against-erc4626-inflation-attacks

Resolution

Peapods Team: Resolved in the following code change.

C-08 | aspTKNOracle Can Be Manipulated With Donation

Category	Severity	Location	Status
Oracle Manipulation	● Critical	aspTKNMinimalOracle.sol	Partially Resolved

Description PoC

In a previous audit, it was reported that the `aspTKN` oracle could be manipulated through a donation of `spTKN` (see https://hackmd.io/@tapir/SyxqzohUA#H-9-aspTKN-oracle-can-be-manipulated).

While this has been fixed through the accounting of assets with a storage variable, this attack is still possible through donation of reward tokens which then gets compounded into `spTKNs` (assets).

`_processRewardsToPodLp` is called on every external user action, which checks for `balanceOf` reward tokens in the contract, and then converts the reward tokens to `spTKNs`.

Similar to the previously reported issue, although the attacker loses the donated tokens, they can manipulate the oracle pricing to borrow more tokens from the lending protocol, exploiting the system. Low liquidity pods are more susceptible to such attacks.

Recommendation

No straightforward solution as existing reward flow relies on transferring tokens to the `AutoCompounder`. Consider re-designing the reward and compounding flow to ensure the oracle pricing cannot be easily manipulated through a donation of reward tokens.

Resolution

Peapods Team: Resolved in the following code change.

Guardian Team: If the intermediate token is also a reward token, then the entire balance would be transferred. This allows an attacker to donate that intermediate token, and it would bypass the `maxSwap` caps that were implemented to remedy the issue originally.

C-09 | Oracle Precision Error Due To Token Decimals

Category	Severity	Location	Status
Arithmetic Error	● Critical	spTKNMinimalOracle.sol: 185	Resolved

Description

The `getPrices` function aims to return price in 18 decimals, which will be consumed by the `aspTKN` oracle and ultimately the `FraxlendPair` contract to determine LTV and borrow amount.

The issue lies in `_calculateBasePerSpTkn` where token decimals are correctly handled up till the calculation of `_pairPrice18`. As the variable name suggests, this is the price of the LP pair returned in 18 decimals:

`uint256 _pairPrice18 = (2 * _avgBaseAssetInLp18 * 10 ** ((_clT0Decimals + _clT1Decimals) / 2)) / IERC20(_pair).totalSupply();`

However, if either token is not in 18 decimals, e.g. USDC: 6 decimals, then the price returned here will not be in 18 `decimals.Assume token0` is 18 decimals and token1 is 6 decimals, the math for decimals works out to be: 18 + ((18 + 6) / 2) - 18 = 12.

The incorrect precision affects all downstream calculations and results in a wrong price consumed by `FraxlendPair`. This ultimately affects LTV calculations which can lead to pairs being drained and users being liquidated unfairly.

Recommendation

Change the calculations to:
`uint256 _pairPrice18 = (2 * _avgBaseAssetInLp18 * 10 ** 18 / IERC20(_pair).totalSupply();`

Afterwards, remove the `_baseTDecimals` logic in `_spTknBasePrice18`.

Resolution

Peapods Team: Resolved in the following code change.

C-10 | UniswapDexAdapter.swapV2Single Doesn't Work

Category	Severity	Location	Status
DoS	● Critical	UniswapDexAdapter.sol	Resolved

Description PoC

`UniswapDexAdapter` uses IUniswapV2Router02.sol to initiate calls to the V2 Router.
In this interface the `swapExactTokensForTokensSupportingFeeOnTransferTokens` functions is expected to return an array with amounts.

However, in the actual implementation of that function there are no values returned. This mismatch will result in a revert every time the function is called causing a DOS for the protocol.

Recommendation

Correct the interface to exclude the returned array.

Resolution

Peapods Team: Resolved in the following code change.

C-11 | Incorrect addInterest Interface

Category	Severity	Location	Status
Integration	● Critical	LendingAssetVault.sol: 245	Resolved

Description

In the function `_updateInterestAndMdInAllVaults`, which is called during every deposit/mint, `addInterest()` is called to trigger interest accrual on the FraxlendPair vault.

However, the wrong interface is used which should pass `bool _returnAccounting` as a function input. Therefore, the current implementation would always fail and DOS all deposits into `LendingAssetVault`.

Recommendation

Use the correct interface for `addInterest`.

Resolution

Peapods Team: Resolved in the following code change.

C-12 | pTKN Share Siphoning Via FlashMint

Category	Severity	Location	Status
Logic Error	● Critical	flashMint()	Resolved

Description

If a smart contract has `IFlashLoanRecipient::callback()` or a `fallback()` function, a malicious user can set them as the receiver of the `flashMint()` function. This will burn .1% of their `pTKN` balance.

Since the only restriction on the amount of `pTKN`s minted is that the total supply does not overflow, this can allow a user to burn the entirety of the contract's pTKN balance.

This is profitable for the malicious user because burning `pTKN`s increases the value of existing `pTKN`s since they are now backed by more underlying tokens.

Recommendation

Charge the fee to the msg.sender instead of the `_recipient`.

Resolution

Peapods Team: Resolved in the following code change.

H-01 | Lack Of Access Control In redeemFromVault

Category	Severity	Location	Status
Access control	High	LendingAssetVault.sol: 324	Resolved

Description

The function `redeemFromVault` can be called by an attacker who passes in an arbitrary `_vault` and `_amountShares`

As this function can be called by anyone, a griefing attack is possible to call `redeemFromVault` to deny `LendingAssetVault` of yield (whenever there is available liquidity in `FraxlendPair`).

Recommendation

Validate the input data and consider only allowing owner to call `redeemFromVault`.

Resolution

Peapods Team: Resolved in the following code change.

H-02 | Whitelist Actions Should Update All Vaults

Category	Severity	Location	Status
Logic Error	High	LendingAssetVault.sol, 267	Resolved

Description PoC

Asset availability changes during `whitelistDeposit` and `whitelistWithdraw`. But, in these functions only the vault that is calling is updated with `_updateAssetMetadataFromVault`. Instead, all whitelisted vaults should be updated too which affects their interest calculations.

Consider this example:
• LAV has 100 DAI
• Two Vaults A & B, with a 100% and 50% max allocation from LAV respectively.
• Vault A & B each have whitelist withdrawn 25 DAI, so A's utilization rate is 25 / 75 = 33% while B's is 25 / 50 = 50%
• 1 day passes
• A new borrower borrows 50 DAI from Vault A, so utilization rate increases from 33 to 100%.
• Available assets are also reduced for vault B, its utilization rate will increase to 25 / 25 = 100%

In the example above, the next time accrued interest in Vault B is calculated, it assumes a 100% utilization rate for the entire duration since the last update.

Instead, it should have been a 50% utilization (lower interest rate) for 1 day and then the 100% utilization rate after the borrow from Vault A. Borrowers will therefore always be incorrectly charged for interest across all whitelisted vaults.

Recommendation

`whitelistDeposit` and `whitelistWithdraw` should update all vaults by calling both: `_updateAssetMetadataFromVault(_vault)` and `_updateInterestAndMdInAllVaults(_vault)`

Resolution

Peapods Team: Resolved in the following code change.

Guardian Team: The recommendation was not implemented.

H-03 | Staking Pool Rewards Sniping Is Possible

Category	Severity	Location	Status
Logic Error	High	TokenRewards.sol	Acknowledged

Description

As there is no penalty nor timelock for unstaking with the `StakingPoolToken`, a user may claim rewards without actually staking by front-running `depositReward` and do: `stake -> depositRewards -> unstake`.

The user would immediately be eligible to claim rewards at the expense of other users who are staked.

Recommendation

Consider implementing a timelock or penalty for unstaking. Also, consider using time-weighted reward distribution.

Resolution

Peapods Team: Acknowledged.

H-04 | Loss Of Rewards Due Two Step Swap Failure

Category	Severity	Location	Status
Logic Error	High	AutoCompoundingPodLp.sol: 381	Resolved

Description

When compounding rewards in `_processRewardsToPodLp`, some rewards may require a two-step process to swap to the paired LP token.

However in `_swapV2`, only a single swap is performed. Therefore, if a second swap is required, it will not be performed and the intermediate token received will remain in the contract.

These rewards will not be compounded and users will lose potential yield. An identical error was found in `Zapper.sol`.

Recommendation

Perform a second swap for tokens which require a two-step process.

Resolution

Peapods Team: Resolved in the following code change.

H-05 | Rewards Are Lost For aspToken

Category	Severity	Location	Status
Logic Error	High	AutoCompoundingPodLp.sol	Resolved

Description

Users stake their LP tokens because `spTokens` accrue rewards in different tokens. The criteria for such a token is to either be a part of the whitelisted ones or be the specified token for the given `TokenRewards` contract.

When `spTokens` are deposited to `aspTokens`, the `AutoCompondingPodLp` contract receives the rewards and uses `_processRewardsToPodLp` to convert them to new `spTokens`. However, it does so only for the whitelisted tokens and ignores the specific reward token for the `TokenRewards`.

In result, any rewards accumulated in the specific token that is not part of the whitelisted tokens will not be correctly distributed to the holders of the `aspTokens`.

Recommendation

In addition to the whitelisted tokens collect the rewards from the `rewardsToken` as well.

Resolution

Peapods Team: Resolved in the following code change.

H-06 | Malicious Function Input In Add/Remove Leverage

Category	Severity	Location	Status
Logic Error	High	LeverageManager.sol	Resolved

Description PoC

In LeverageManager, the addLeverage and removeLeverage functions allows an attacker to pass in a malicious contract as _selfLendingPairPod and _dexAdapter respectively.

This opens up the surface for attacks and reentrancy. It could be used for example to avoid payment of close fees during removeLeverage:

1. Alice calls removeLeverage passing in malicious contract as _dexAdapter
2. Malicious contract is called in _swapPodForBorrowToken
3. Malicious contract does the swap with a DEX but does not return any pod tokens, instead transferring directly to Alice
4. As no pod tokens remain in LeverageManager, no close fees are applied at the end of callback and the protocol loses revenue.

Recommendation

Perform validation on the _selfLendingPairPod and _dexAdapter inputs. Do not allow users to provide arbitrary dexAdapter addresses, and use every WeightedIndex's own immutable dexAdapter.

Additionally, consider minimizing the use of arbitrary inputs in public functions, as they can expand the attack surface and introduce potential vulnerabilities, such as reentrancy risks

Resolution

Peapods Team: Resolved in the following code change.

Guardian Team: Because there is no validation on _overrideLendingPair, users can pass in an arbitrary, malicious contract for this address. The malicious contract can return zero for the _podAmtRemaining, which the protocol relies upon to calculate the _closeFeeAmt.

H-07 | Reward Sniping With AutoCompounder

Category	Severity	Location	Status
Logical Error	High	AutoCompoundingPodLp.sol	Acknowledged

Description

When token rewards such as ARB are deposited into the `TokenRewards` contract, a large portion of it is expected to be transferred to `AutoCompoundingPodLp` which is a large holder of `spTKNs`.

Then the next time a user interacts with `AutoCompoundPodLp`, `_processRewardsToPodLp` is called to compound the reward tokens into more LP, benefiting existing holders of `aspTKN`.

Recommendation

As there is no penalty nor timelock for deposits and withdrawals, a user may front-run the deposit of reward tokens by depositing into `AutoCompoundingPodLp` so as to claim some of the rewards, and then back-run the deposit of rewards and withdraw from `AutoCompoundingPodLp`. Such extractive behavior results in less rewards for other users who are staked for the long term.

Resolution

Peapods Team: Acknowledged.

H-08 | removeLeverage Inaccurate Share Calculation

Category	Severity	Location	Status
Logic Error	High	LeverageManager.sol: 169	Resolved

Description

When removing leverage, the amount of shares to repay to the Fraxlend vault is calculated based off the amount of the borrowed assets that are desired to be repaid. This is accomplished by calling `toShares()`.

However, this calculation is done before any call is made to the Fraxlend vault. Because of this, the calculation of shares to be repaid does not account for any interest that has been accrued.

If enough time has passed or the interest rate is high enough, it will lead to an inaccurate amount of shares to pay off based on the amount of assets provided. This will lead to a revert in `repayAssets()`, due to insufficient balance and DoS `removeLeverage()`

Recommendation

Prior to the calculation of `_borrowSharesToRepay` in `removeLeverage()`, call `addInterest()` on the vault.

Resolution

Peapods Team: Resolved in the following code change.

H-09 | Liquidators Can Avoid Bad Debt Socialization

Category	Severity	Location	Status
Logic Error	High	FraxlendPair.sol	Resolved

Description PoC

During liquidation of a borrower, bad debt is only realized when the borrower has zero leftover collateral (see FraxlendPairCore.sol: 1130). This allows a liquidator to liquidate just enough shares such that a dust amount of collateral is left behind.

Thereafter, there might be little to no incentive for other liquidators to liquidate the borrower as the gas cost to do so exceeds the collateral value.

As a result, the bad debt is not socialized and lenders may exit the system without any losses. The liquidator himself may be a lender and therefore incentivized to exploit this loophole.

Recommendation

Consider implementing a threshold for collateral remaining after liquidation, such that a liquidator must leave sufficient collateral behind if doing a partial liquidation.

Resolution

Peapods Team: Resolved in the following code change.

H-10 | DoS In _withdrawToVault Due To Underflow

Category	Severity	Location	Status
DoS	High	FraxlendPairCore.sol: 1012	Resolved

Description PoC

When the Fraxlend pair does not have enough assets to lend, necessary amounts are transferred from the LendingAssetVault(LAV), and Frax shares are minted to LAV. The opposite occurs when removing leverage: Frax shares are burned from the LAV, and assets are transferred back to the vault.

The share amounts to mint and burn are always in favor of the protocol. Frax shares that the LAV receives during borrowing are rounded down in _depositFromVault, while Frax shares burned during repayment are rounded up in _withdrawToVault, as expected.

In the _repayAsset function, the _withdrawToVault is called with the asset amounts to repay. However, this causes DoS in certain situations. When attempting to transfer the entire utilized amount back (_extAmount == _externalAssetsToWithdraw), the share amount is rounded up in _withdrawToVault, causing the function to revert due to the insufficient balance error, as the LAV holds 1 fewer shares.

Recommendation

Check the share balance of the LAV before burning, and burn shares up to LAV balance.

Resolution

Peapods Team: Resolved in the following code change.

H-11 | Assets Can Be Borrowed/Repaid While Paused

Category	Severity	Location	Status
Logic Error	High	FraxlendPairCore.sol	Resolved

Description

`borrowAsset` and `repayAsset` functions may be paused by admin but can be bypassed through `leveragePosition` and `repayAssetWithCollateral` which performs borrow/repay actions. This loophole could be exploited by attackers while the protocol is paused.

Recommendation

Consider extending the pause effects to the `leveragePosition` and `repayAssetWithCollateral` functions.

Resolution

Peapods Team: Resolved in the following code change.

H-12 | Lack Of _selfLendingPairPod Validation

Category	Severity	Location	Status
Validation	High	LeverageManager.sol: 97	Resolved

Description

When calling `addLeverage()`, a user is allowed to input whatever `_selfLendingPairPod` they desire, without any validation. This will allow a user to successfully add leverage with a self lending pod that is different from the pod they used to create their position.

However when they attempt to withdraw, they will be forced to use the pod associated with their NFT. This will prevent a user from removing leverage on their position, and force the position to be open indefinitely.

Since positions are transferable, a malicious user could sell their position to an unsuspecting user. This will lead to a user being stuck with a worthless position.

Recommendation

Validate that the `_selfLendingPairPod` passed in to `addLeverage()` is the same pod associated with their NFT. Alternatively, allow users to update the `selfLendingPod` they have associated with their position NFT.

Resolution

Peapods Team: Resolved in the following code change.

H-13 | Flash Mint Manipulates Supply

Category	Severity	Location	Status
Logical Error	High	DecentralizedIndex.sol	Resolved

Description

`WeightedIndex.flashMint()` allows anyone to sandwich protocol actions by manipulating `totalSupply()`.

There are a lot of parts in the protocol that depend on `totalSupply`:

• `WeightedIndex.convertToShares()`

• `WeightedIndex.convertToAssets()`

• `ConversionFactorPTKN._calculateCbrWithDen()`

Recommendation

Either change the code that depends on `totalSupply` or reconsider the existence of the `flashMint` function.

Resolution

Peapods Team: Resolved in the following code change.

M-01 | Insufficient Token Amounts Lead To Swap Error

Category	Severity	Location	Status
Logic Error	Medium	AutoCompoundingLp.sol	Partially Resolved

Description

When compounding rewards in `_pairedLpTokenToPodLp`, `pairedLpTokens` are swapped to pTKN via Uniswap's `swapV2Single`. However, if too little tokens are provided it may revert in Uniswap V2 with 'INSUFFICIENT_INPUT_AMOUNT' or `INSUFFICIENT_OUTPUT_AMOUNT` .

This could be caused by a balance of 1 wei of `pairedLpTokens`. which after halving becomes 0 pTKNs. This small balance could be easily donated by an attacker looking to DOS the contract, or simply caused by leftover tokens from a previous transaction.

As processing of rewards is called by every major flow, reverting could have serious implications such as preventing users from removing leverage and result in liquidations.

Recommendation

Verify the balance of tokens before calling the swap function to avoid reverts.

Resolution

Peapods Team: Resolved in the following code change.

M-02 | Underflow In _updateAssetMetadataFromVault

Category	Severity	Location	Status
Arithmetic Error	Medium	LendingAssetVault.sol: 305	Partially Resolved

Description PoC

Whenever `_updateAssetMetadataFromVault` is called, a vault's Collateral Backed Ratio (CBR) is updated and compared against its previous value. If the CBR decreased from the previous update, then the vault's utilization is also decreased based on `_vaultAssetRatioChange`.

The issue occurs when `_vaultAssetRatioChange` is greater than 100%. This leads to an underflow when updating `vaultUtilization[_vault]`.

Consider this example:
• Vault's CBR decreased from 100e27 to 49e27
• `_vaultAssetRatioChange`: (100e27 * 1e27 / 49e27) - 1e27 = 1.04e27
• `vaultUtilization[_vault]`: 100 - (100 * 1.04e27 / 1e27) = underflow

Such a drastic drop in CBR is unlikely but possible in vaults with obscure tokens (as Peapods is designed to be used permissionlessly). A revert in the update for one vault will cause DOS in all whitelisted vaults, and prevent liquidations in `FraxlendPair` vaults.

Recommendation

Handle the case when `_vaultAssetRatioChange` is greater than 100% to avoid the underflow.

Resolution

Peapods Team: Resolved in the following code change.

M-03 | Precision Loss Leads To Reverts In TokenRewards

Category	Severity	Location	Status
Precision	Medium	TokenRewards.sol: 365, 253-256, 269-273, 372	Resolved

Description

When a user receives shares from `TokenRewards` for the first time, `_cumulativeRewards` is called to update the user's rewards mapping with an `excluded` amount. This amount will be used to calculated the user's share of future rewards.

The issue occurs in `_cumulativeRewards:` `(_share * _rewardsPerShare[_token]) / PRECISION`

By rounding down the calculation, it is essentially excludes the user from 1 wei less of rewards, implying the user earned 1 wei more rewards. This will overtime lead to insufficient balance of rewards to transfer out, and DOS all functionality of the contract when that happens.

Recommendation

Always round up the calculation in `_cumulativeRewards` when calculating the excluded amount.

Resolution

Peapods Team: Resolved in the following code change.

M-04 | Positions Are Lost When lendingPair Is Changed

Category	Severity	Location	Status
Logical Error	Medium	LeverageManager.sol	Resolved

Description

When a new position is initialized, its lendingPair is set to the lending pair for the pod configured by the owner of the contract. However, adding and removing leverage always use the most recent `lendingPair` for the pod of the position instead of the pair at the time of it creation.

This results in positions being lost when the owner changes the pair by calling setLendingPair because _removeLeverage will make the custodian remove collateral from the new pair where it doesn't have any.

Recommendation

Use `positionProps.lendingPair` instead of the latest lending pair when adding/removing leverage.

Resolution

Peapods Team: Resolved in the following code change.

M-05 | Flash Loan Repayments Fail During addLeverage

Category	Severity	Location	Status
Logical Error	Medium	LeverageManager.sol: 327-329	Resolved

Description

When adding leverage, the user provides pod tokens, and the corresponding `pairedLpTokens` are flash loaned. At the end of the transaction, this flash loan is repaid by borrowing `pairedLpTokens` from Fraxlend.

However, the flash loan fee is not accounted for when borrowing from Fraxlend. The borrow amount from Fraxlend equals the flash loan amount unless users provide a greater `overrideBorrowAmt`.

The Natspec comments regarding this variable is: ”Override amount to borrow from the lending pair, only matters if max LTV is >50% on the lending pair”.

Since providing `overrideBorrowAmt` is not mandatory and there are no restrictions, the borrow amount from Fraxlend will usually be equal to `_props.pairedLpDesired` in most cases. However, this amount is equal to `_d.amount`, which is smaller than `_flashPaybackAmt`, causing the transaction to revert.

Recommendation

The minimum borrow amount from Fraxlend to repay the flash loan should be `_props.pairedLpDesired + _d.fee`.

Resolution

Peapods Team: Resolved in the following code change.

M-06 | Borrowers Pay For Paused Interest

Category	Severity	Location	Status
Logical Error	Medium	Fraxlend.sol	Resolved

Description

`FraxlendPair` has a function pause which pauses all actions in the pair and a function `pauseInterest` which pauses interest accrual.

When interest is paused, new interest will not be accumulated and that's expected. However, once unpaused, the current borrowers will have to pay interest for the duration from the moment the protocol was paused until the current block.

Since the protocol may function normally and have just it interest paused, that means new lenders and borrowers may come and go. This will cause a huge interest misaccounting.

For example, interest is paused on Monday. Some borrowers leave the pair. and new ones enter it right before it's unpaused the next Monday. Since the `lastUpdated` timestamp will be the first monday, the new borrowers will immediately owe interest for that one week they were not even part of the pair.

Recommendation

Update the timestamp when `pauseInterest`(false) is called `currentRateInfo.lastTimestamp = uint64(block.timestamp);`

Resolution

Peapods Team: Resolved

M-07 | Improper Slippage And Deadline During Deposit

Category	Severity	Location	Status
Logic Error	Medium	AutoCompoundingPodLp.sol: 102	Acknowledged

Description

When calling `deposit`, a slippage of `0` and a deadline of `block.timestamp` is passed to `_processRewardsToPodLp`. Using `0` for slippage is dangerous as MEV bots could sandwich the swap to steal tokens.

Similarly, setting block.timestamp as deadline is ineffective as the transaction could sit in the mempool until it's ready to be processed, at which time block.timestamp is set, therefore offering no protection from sandwich attacks.

Recommendation

Allow user to input slippage and deadline parameters when depositing.

Resolution

Peapods Team: Acknowledged.

M-08 | AutoCompoundingPodLp Is Not EIP Compliant

Category	Severity	Location	Status
ERC4626	Medium	AutoCompoundingPodLp.sol	Resolved

Description

Some EIP-4626 compliance issues have been observed in the AutoCompoundingPodLp contract:
1. According to EIP-4626, the mint function must mint exactly the user-inputted amount of shares, and the withdraw function must transfer exactly the specified assets amount. However, these functions mint or withdraw fewer tokens due to rounding down twice during the action flows.

During the withdraw function in the codebase:
• User provides _assets amount.
• It is converted to shares with convertToShares function, which rounds down.
• Then, the internal _withdraw function is called with this shares amount.
• In this internal function, the shares amount is converted to assets again with convertToAssets, which also rounds down.

As a result, the actual assets amount transferred to the user is not the same as the user-provided amount. The same issue can be observed in the mint function as well.
1. According to EIP-4626, withdraw and redeem functions must support transaction flows where the msg.sender has an approval from the owner. However, in the codebase, withdrawals and redeems can only be performed by the owner, and approved users cannot execute these actions.

Recommendation

Update mint and withdraw functions to comply with the EIP specification and ensure the exact amounts are transferred. Also, update withdraw and redeem function to support approved users.

Resolution

Peapods Team: Resolved in the following code change.

M-09 | Rewards Not Updated Prior To Fee Change

Category	Severity	Location	Status
Logic Error	Medium	AutoCompoundingPodLp.sol	Resolved

Description

Owner can set a new protocol fee via `setProtocolFee`. However, because rewards are not updated prior to the fee change, the new fee will apply to previously accrued rewards.

For example, 100 PEAS in rewards were accrued since the last update. Fees are increased from 1 to 2%. An additional 1% of fees are unjustly applied to the accrued rewards.

Recommendation

In `setProtocolFee`, call `_processRewardsToPodLp` before setting `protocolFee` to the new fee.

Resolution

Peapods Team: Resolved in the following code change.

M-10 | Vault Whitelist Can Be Set One Above Max

Category	Severity	Location	Status
Logic Error	Medium	LendingAssetVault.sol: 360	Resolved

Description

When whitelisting a new vault with `setVaultWhitelist`, even if `maxVault` value has been reached, the new vault is still added to the whitelist due to using `<= maxValue` instead of "`< maxValue`" in the check.

Recommendation

Change from `require(_vaultWhitelistAry.length <= maxVaults, 'M');`
to `require(_vaultWhitelistAry.length < maxVaults, 'M');`

Resolution

Peapods Team: Resolved in the following code change.

M-11 | Inflated Admin Fee

Category	Severity	Location	Status
Logic Error	Medium	TokenRewards.sol: 301	Resolved

Description

When a swap fails in `depositFromPairedLPToken()`, the amount that can be used in the next attempt is halved from the attempted swap amount.

The next time `depositFromPairedLPToken()` is called the fee will be based off the current balance of the contract, which will include the balance from the failed swap.

When `_swapForRewards` is called, it will reduce the `_amountIn` and `_amountOut` but still charge the admin fee based on the balance of the contract. This results in excessive admin fees paid over the multiple swaps.

Recommendation

Adjust the admin fee to match the actual amount that has been swapped.

Resolution

Peapods Team: Resolved in the following code change.

M-12 | Same Heartbeat For Multiple Oracles

Category	Severity	Location	Status
Oracles	Medium	ChainlinkSinglePriceOracle.sol, 137-144	Resolved

Description

`getPriceUSD18` makes requests to a base and quote asset oracles. If any of them has been updated more than `maxOracleDelay` seconds ago, `_isBadData` will be set to `true`.

Since not all feeds have the same heartbeat, if the oracle uses two feeds with different ones, it may happen that one of the prices is stale, but it's accepted as a valid one.

Recommendation

Use two different delay variables - one for the base feed and one for the quote feed.

Resolution

Peapods Team: Resolved in the following code change.

M-13 | No Circuit Breaker Checks In ChainlinkOracle

Category	Severity	Location	Status
Oracles	Medium	ChainlinkSinglePriceOracle.sol	Resolved

Description

`getPriceUSD18` returns `_isBadData` if the oracle price is stale. However, it doesn't consider the price going outside of the price range for the oracle's aggregator.

The price will be capped between `minAnswer` and `maxAnswer` of the aggregator. This will result in a wrong price being used in the protocol.

Even though most feeds have disabled their circuit breaker feature, there are still some that haven't, for example CVX/ETH

Recommendation

If the price goes outside the aggregator range, set `_isBadData` to true

Resolution

Peapods Team: Resolved in the following code change.

M-14 | DOS Of Borrow & Redeem

Category	Severity	Location	Status
Logic Error	Medium	FraxlendPairCore.sol: 611	Resolved

Description

Each time borrow or redeem is called in FraxlendPairCore, if there are insufficient local assets, then _depositFromVault is called to pull assets from the vault.

However, in _depositFromVault there is a check:
if (depositLimit < _totalAsset.totalAmount(address(externalAssetVault))) revert ExceedsDepositLimit();

This check prevents the deposit from vault if the vault's allocated assets to the FraxlendPair exceeds the deposit limit.

Consider this example:
• FraxlendPair has a deposit limit of 10 ETH
• Vault has 30 ETH and allocates 50% to FraxlendPair
• Borrow/redeem actions cannot go through as the depositLimit check would always fail

Recommendation

Consider removing the depositLimit check from _depositFromVault. Instead, in LendingAssetVault, apart from percentage based allocations to a FraxLendPair vault, consider checking for the vault's depositLimit too to avoid over-allocation.

Resolution

Peapods Team: Resolved in the following code change.

M-15 | Feeds With > 18 Decimals Are Problematic

Category	Severity	Location	Status
Math	Medium	ChainlinkSinglePriceOracle.sol	Resolved

Description

The price returned by the oracle is adjusted to 18 decimals with the following computation: `_price18 = uint256(_price) * (10 ** 18 / 10 ** _decimals);`

Notice that the division here happens before the multiplication. This means that if the decimals of the feed are more than 18, the price will be rounded to 0 causing big problems for the assets pricing.

Recommendation

Multiply price by 1e18 and divide afterwards.

Resolution

Peapods Team: Resolved in the following code change.

M-16 | addLiquidity Fails For Fee-On-Transfer Tokens

Category	Severity	Location	Status
Logic Error	Medium	DecentralizedIndex.sol	Acknowledged

Description

When `addLiquidityV2` is called, an amount of `_pairedLPToken` is transferred into the contract, and the same amount is used to add liquidity in the `DEX_HANDLER`.

However, if the `pairedLPToken` is a Fee-on-Transfer token, then the amount received would be less than expected due to a fee. Therefore, the `DEX_HANDLER.addLiquidity` call could fail due to insufficient tokens.

Recommendation

Use actual balance of `pairedLPTokens` when calling `DEX_HANDLER.addLiquidity`.

Resolution

Peapods Team: Acknowledged.

M-17 | Single Token Pod Assumption

Category	Severity	Location	Status
Logic Error	Medium	spTKNMinimalOracle.sol: 206	Acknowledged

Description

When getting price from the oracle, if the base token is a pod, `_getBaseTokenInClPool` is called. There it gets all assets from the pod and assumes the first token in the array is the base token.

However, pods were designed to be multi-asset and able to be created permissionlessly. Therefore, if the first asset is not the intended base token, then serious integration issues would occur.

Furthermore in the function `_debondFromSelfLendingPod`, an assumption is made that the `selfLendingPod` has only one token. If ever this assumption is broken, there would be integration errors with `FraxlendPair` and a possibility of stuck tokens in `LeverageManager` after debonding.

Recommendation

In the constructor, similar to how `UNDERLYING_TKN` is defined, store the intended underlying token for the base (pod). Furthermore, consider handling the case where a `SelfLendingPod` has multiple tokens.

Resolution

Peapods Team: Acknowledged.

M-18 | Wrong Price Calculations If T0 Is baseToken

Category	Severity	Location	Status
Protocol	Medium	spTKNMinimalOracle.sol	Resolved

Description

When the base token is one of the two tokens in the `UNDERLYING_TKN_CL_POOL`, it has to be `token1`. That's because `_pricePTKNPerBase18` is calculated based on whether or not the base token is part of that pool by checking if it's `token1`.

This means for pools where the base token is `token0` the price will be wrongly flipped.

Recommendation

If the base token is included in the pool pair, always make sure it's the `token1`.

Resolution

Peapods Team: Resolved.

M-19 | Improper Deadline For Fraxlend Swaps

Category	Severity	Location	Status
Logic Error	Medium	FraxLendPairCore.sol: 1341	Resolved

Description

Similarly to M-01, `FraxlendPairCore::repayAssetWithCollateral()` & `FraxlendPairCore::leveragedPosition()` does not allow a user to set the `block.timestamp` for their swap.

This exposes users to MEV sandwich attacks, and can cause them to lose out on funds that would have been used to repay their debt.

Recommendation

Allow users to input the deadline for the swaps.

Resolution

Peapods Team: Resolved in the following code change.

M-20 | FraxVault Incompatible With Non-Standard Tokens

Category	Severity	Location	Status
Logical Error	Medium	FraxlendPair.sol	Acknowledged

Description

The FraxlendPair contracts do not support non-standard tokens such as rebasing or fee-on-transfer tokens, whose balance changes during transfers or over time. If the Peapods team expects to support these tokens, then there will be accounting issues when interacting with FraxlendPair contracts.

Recommendation

Verify the amount of tokens transferred to the contracts before and after the actual transfer to infer any fees/interest.

Resolution

Peapods Team: Acknowledged.

M-21 | removeLeverage Could Fail For Self-Lending Pairs

Category	Severity	Location	Status
Logical Error	Medium	LeverageManager.sol	Resolved

Description

This bug was reported in a previous audit but does not seem to be fixed (see https://hackmd.io/@tapir/SyxqzohUA#M-6-Removing-leverage-will-likely-fail-if-the-pod-token-needs-to-be-sold-for-the-borrowed-token-in-a-self-lending-scenario)

The issue remains that there is no natural Uniswap V2 market that exists to swap pod tokens for borrowed assets (from `FraxlendPair`), when the pair is self-lending.

Furthermore, during `_swapPodForBorrowToken` in the `removeLeverage` flow, the entire amount of pod tokens received is passed as `amountInMax` for the swap --resulting in zero slippage protection.

This could allow an attacker to deploy a pool to take advantage of this scenario and steal all pod tokens from a user.

Recommendation

Implement proper slippage protection for the swap, and ensure that a healthy Uniswap V2 market exists for the swapping of self-lending pairs.

Resolution

Peapods Team: Resolved in the following code change.

M-22 | USDC Blacklist Prevents Transfers

Category	Severity	Location	Status
Logic Error	Medium	TokenRewards.sol: 247, 24-28, 247-249	Resolved

Description

When the staking pool token is transferred it will call `_setShares()`, which will lead to `_distributeReward()` being called.

Inside of `_distributeReward()`, it will loop through the reward tokens and transfer any rewards to the users. If a user becomes blacklisted from using USDC, `_distributeReward()` will revert.

This, in turn, will lead to the tokens being stuck in the users wallet and become untransferable. Additionally, this prevents a user from calling `claimRewards()`. They will not be able to claim any other rewards tokens earned outside of USDC.

Recommendation

Instead of pushing rewards to users automatically, rely on them claiming the rewards themselves and allow them to specify which token they would like to claim.

Resolution

Peapods Team: Resolved in the following code change.

M-23 | Uniswap Ticks Rounding

Category	Severity	Location	Status
Math	Medium	Global, 95-102, 99-106	Resolved

Description

Multiple contracts in the system -V3TwapUtilities, V3AerodromeUtilities, UniswapV3SinglePriceOracle - fetch the TWAP price of a given asset. When calculating the TWAP, the delta of two CL ticks is divided by a given time period.

Since solidity truncates when it divides, for negative ticks the result will be rounded up instead of rounded down resulting in a different price. For reference, see how is this handled in OracleLibrary.

Recommendation

Implement the same solution as in `OracleLibrary`

Resolution

Peapods Team: Resolved in the following code change.

M-24 | Arbitrage From Deviation In Oracle Price

Category	Severity	Location	Status
Oracles	Medium	FraxlendPair.sol	Acknowledged

Description

The maximum a user can borrow is determined by the exchange rate returned by the oracle. However, all oracles are susceptible to front-running as their prices tend to lag behind an assets real price.

For example, Chainlink oracles are updated after price crosses a threshold while Uniswap V3 TWAP returns price over past X blocks. An attacker could exploit the difference between the price reported by an oracle and the asset's actual price to gain a profit by front-running the oracle's price update.

The likelihood of this condition is increased for Peapods due to the multiple layers that an underlying asset is wrapped in.

Consider this example:
• Fraxlend Vault has a high `maxLTV` of 95%, with the collateral as `aspTKN` (`pPEAS-WETH`) and asset (WETH)
• 1 `aspTKN` is currently worth 0.002 WETH
• Price of `aspTKN` drops while WETH price increases, such that 1 `aspTKN` should be worth 0.0015 WETH
• Due to the lag in oracle update, price is not updated yet
• Attacker sees this opportunity and front-runs the oracle update to:
• Deposit 100 `aspTKN`s
• Max borrow 0.19 WETH (95% LTV)
• Afterwards, the oracle price is updated to 1 `aspTKN` = 0.0015 WETH
• The attacker's position is now unhealthy as his collateral is worth less than the loan amount
• Attacker back-runs the oracle update to liquidate himself:
• To seize 100 `aspTKN` he repay 0.15 WETH
• Gains back his original collateral plus 0.04 WETH

All profits gained result in bad debt socialized among lenders.

Recommendation

Consider adding a borrowing fee to mitigate arbitrage opportunities.

Resolution

Peapods Team: Acknowledged.

M-25 | mint In Fraxlend Rounds In Users' Favour

Category	Severity	Location	Status
Rounding	Medium	FraxlendPairCore.sol: 660	Resolved

Description

The `mint` function in the `FraxlendPairCore` contract rounds down in favour of the users and users pay fewer assets for corresponding shares.

Recommendation

Round-up in favour of the protocol.

Resolution

Peapods Team: Resolved in the following code change.

M-26 | Oracle Incompatible With Non-Standard Tokens

Category	Severity	Location	Status
Underflow	Medium	spTKNMinimalOracle.sol: 104	Resolved

Description

The current logic in getPrices will underflow when the BASE token has more than 18 decimals: `uint256 _priceOne18 = _priceBaseSpTKN * 10 ** (18 - IERC20Metadata(BASE_TOKEN).decimals());`

This will entirely prevent oracle compatibility with borrow tokens that have more than 18 decimal precision, which is problematic in Peapods which is a permissionless system.

Recommendation

Query the decimals first and if it is greater than 18, subtract 18 from the base token's decimals.

Resolution

Peapods Team: Resolved in the following code change.

M-27 | Fee-on-transfer Tokens Are Not Supported

Category	Severity	Location	Status
Logic Error	Medium	Global	Acknowledged

Description

Fee on transfer tokens are not supported correctly for:

• pod underlying token - `bond()` doesn't check the received amount of the transfer and mints shares based on the initial amount

• pod paired token - `LeverageManager` assumes it has received the whole amount of the flashloaned token

Recommendation

Consider supporting fee on transfer tokens

Resolution

Peapods Team: Acknowledged.

M-28 | Missing Check For Sequencer Downtime

Category	Severity	Location	Status
Oracle Manipulation	Medium	ChainlinkSinglePriceOracle.sol	Resolved

Description

Chainlink recommends that all Optimistic L2 oracles consult the Sequencer Uptime Feed to ensure that the sequencer is live before trusting the data returned by the oracle.

See https://docs.chain.link/data-feeds#l2-sequencer-uptime-feeds

If the Arbitrum sequencer goes down for example, oracle data will not be updated and could become stale. Attackers could take advantage of the stale prices and carry out attacks, such as borrowing more against their collateral's true value.

Recommendation

Follow the code example of Chainlink:
https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code

Resolution

Peapods Team: Resolved in the following code change.

M-29 | ASP Insufficient Liquidity DOS

Category	Severity	Location	Status
DoS	Medium	AutoCompoundingPodLp.sol	Resolved

Description

When `AutoCompoundingPodLp` swaps the paired tokens for pod tokens, it adds them as liquidity. However, if the amounts are too small, this will result in minting 0 liquidity and a revert with `INSUFFICIENT_LIQUIDITY_MINTED`.

Recommendation

Just like in the `DecentralizedIndex`, consider rewards only if they exceed a given minimum.

Resolution

Peapods Team: Resolved in the following code change.

M-30 | User's Lockup Period Should Not Change Midway

Category	Severity	Location	Status
Logic Error	Medium	VotingPool.sol, 59	Resolved

Description

Owner can set the `lockupPeriod` variable via `setLockupPeriod`. However, this would affect all users who are already staked with the previous `lockupPeriod` value, which would be unfair.

Recommendation

When a user stakes, consider storing the current `lockupPeriod` value in their own `Stake` struct. And use that value when checking for unlock time in `unstake`.

Resolution

Peapods Team: Resolved in the following code change.

M-31 | Asp Rewards Can Be Sandwiched

Category	Severity	Location	Status
Logic Error	Medium	AutoCompoundingPodLp.sol	Resolved

Description

The idea of AutoCompoundingPodLp is to swap the accumulated rewards into paired lp tokens and use them to generate new spTokens.

When the current reward token of the asp doesn't match the reward token for its pod, 0 slippage is used for the swap so anyone can sandwich the transaction to benefit from it which will result in a loss for the asp holders.

The following can be executed in one transaction:
• swap
• process rewards
• swap again

Recommendation

Consider adding an adequate slippage parameter to the swaps and a try/catch as well to not introduce a new way of DOS-ing the asp.

Resolution

Peapods Team: Resolved.

M-32 | Incorrect Autocompounding Asset And Shares Conversions

Category	Severity	Location	Status
Logic Error	Medium	AutoCompoundingPodLp.sols: 134 & 154	Resolved

Description

In `AutoCompoundingPodLp::withdraw()`, it will convert the amount of assets to shares prior to calling `_processRewardsToLp()`. Then it will convert the shares back to assets.

In between conversions, the `_cbr()` is likely to increase due to the increase in total assets that will occur when `_processRewardsToLp()` is called. This will lead to a larger output amount of assets than requested to be withdrawn.

When `AutoCompoundingPodLp::mint()` is called, it will convert the amount of shares to assets. Then call `_processRewardsToLp()`, and proceed to convert the amount of assets back to shares.

This will have the inverse effect, and provide a smaller amount of shares for the deposited user than requested. Ultimately, both of these functions will provide users with different amount of shares and assets respectively than expected.

Recommendation

`_processRewardsToLp()` should be called in the beginning before any conversions.

Resolution

Peapods Team: Resolved.

Guardian Team: In functions `withdraw` and `redeem` asset-share calculations are made with a stale cbr since conversions are made before calling `_processRewardsToLp()`. This will lead to incorrect outputs for the user.

M-33 | LendingAssetVault Asset/Share Conversion Error

Category	Severity	Location	Status
Logic Error	Medium	LendingAssetVault.sol: 149 & 169, 197	Resolved

Description

Similarly to M-20, the `LendingAssetVault::withdraw()` and `LendingAssetVault::mint()` perform asset and share conversions with an update to `_cbr()` taking place in between.

This takes place with the call to `_updateInterestAndMdInAllVaults()` happening in `_withdraw()` and `_deposit()`. This will lead to a similar scenario where the accounting for assets will be incorrect for withdrawals and the shares will be incorrect for mints compared to the amounts requested.

Recommendation

`_updateInterestAndMdInAllVaults()` should be called in the beginning rather than between conversions.

Resolution

Peapods Team: Resolved in the following code change.

Guardian Team: LendingAssetVault mint performs `convertToAssets` before updating for interest. This will lead to users minting not getting the amount of shares requested, which is against ERC4626 spec and unexpected for users.

M-34 | VotingPool Pods Priced Equally

Category	Severity	Location	Status
Protocol	Medium	VotingPool.sol	Acknowledged

Description

Currently, all pods are priced equally in the VotingPool contract (assuming the conversion factor is 1:1). This means users can stake cheap pods and receive the same amount of voting tokens they would have received with more expensive pods.

For example, a pod with DAI as underlying token and a pod with WETH as underlying token would both be priced equally if their conversion factors are the same.

There may also be a situation where the pod with DAI token has its conversion factor higher - this will lead to the DAI pod minting more voting tokens than the WETH one.

Recommendation

Either implement another pricing mechanism or make sure to only use pods with close price.

Resolution

Peapods Team: Acknowledged.

M-35 | VotingPool Incompatible With Non-Standard Tokens

Category	Severity	Location	Status
Integration	Medium	VotingPool.sol	Acknowledged

Description

If the underlying token of a pod is a Fee-on-Transfer token, the accounting when staking in `VotingPool` would be inaccurate. The balance of tokens after fees should be accounted for instead. Multi-asset pods are also incompatible as `_calculateCbrWithDen` assumes `_asset[0]` is the only token in the pod.

Recommendation

As Peapods is expected to be permissionless and work with all types of tokens, handle such non-standard tokens accordingly.

Resolution

Peapods Team: Acknowledged.

M-36 | redeemFromVault DOS

Category	Severity	Location	Status
DoS	Medium	LendingAssetVault.sol	Resolved

Description

When `LAV.redeemFromVault()` is called, `FraxlendPair.redeem()` is called and the returned value (the assets received) are subtracted from the `vaultUtilization`.

Since `vaultUtilization` is adjusted by dividing in `_updateAssetMetadataFromVault`, `vaultUtilization` may end up being 1 wei less than the received assets. Because of this the `redeemFromVault` transaction will fail.

Recommendation

Subtract the minimum between the received assets and `vaultUtilization`.

Resolution

Peapods Team: Resolved in the following code change.

M-37 | Vaults' Utilization Not Updated After Bad Debt

Category	Severity	Location	Status
Logic Error	Medium	FraxlendPairCore.sol: 1135	Resolved

Description

During liquidation in `FraxlendPair`, if bad debt was incurred, `whitelistUpdate` is called which updates of all whitelisted vaults (except the calling vault).

Then the bad debt is realized via: `totalAsset.amount -= _amountToAdjust` before `LendingAssetVault` is updated again to reduce its own internal tracking for `_totalAssets`.

Instead, the `whitelistUpdate` of all vaults should be performed after the bad debt is realized in `FraxlendPair`. As a result, all other vaults will assume a higher amount of available assets (did not account for lost assets from bad debt) and charge a higher interest.

Recommendation

Call `whitelistUpdate` at the end of liquidate after all state changes have been made in `FraxlendPair`

Resolution

Peapods Team: Resolved in the following code change.

M-38 | Same TWAP For Multiple V3 Pools

Category	Severity	Location	Status
Oracles	Medium	spTKNMinimalOracle.sol	Acknowledged

Description

`spTKNMinimalOracle` uses the same `twapInterval` for two different pools. Depending on the available liquidity on the two pools and the assets volatility, one twap period may not be sufficient to get accurate prices for both pools.

Recommendation

Consider having a different interval for each pool.

Resolution

Peapods Team: Acknowledged.

M-39 | _getPairedTknAmt 0 Bond Slippage

Category	Severity	Location	Status
Logic Error	Medium	LeverageManager.sol	Acknowledged

Description

`LeverageManager._getPairedTknAndAmt` uses 0 as slippage parameter when bonding tokens. In result, the received `pToken` amount may be too small that it causes significant loss for the user.

Recommendation

Allow the user to input their slippage.

Resolution

Peapods Team: Acknowledged.

M-40 | getPairAccounting Includes LAV Assets

Category	Severity	Location	Status
Logic Error	Medium	FraxlendPair.sol	Resolved

Description

FraxlendPair.getPairAccounting includes the unlent assets from the LAV. External integrations that depend on that function will receive wrong information.

For example, if they calculate the value of a single share using the output of that function, their result will be wrong because they account for assets not present in the pair.

Recommendation

Exclude the LAV assets.

Resolution

Peapods Team: Resolved in the following code change.

M-41 | Donation Increases Share Supply

Category	Severity	Location	Status
Logical Error	Medium	LendingAssetVault.sol	Resolved

Description PoC

Function `donate` aims to increase the `totalAssets` of the `LendingAssetVault` without increasing the `totalSupply` of shares, hence a donation.

The issue is that `_burn(address(this), convertToShares(_assetAmt));` converts the `_assetAmt` to shares after `_deposit(_assetAmt, address(this));` already minted shares, so the newly calculated share amount to burn will be less than the calculated and minted shares in `_deposit`.

Ultimately, function `donate` increases the `totalSupply` even though it is not meant to.

Recommendation

Burn the entire added supply post-deposit.

Resolution

Peapods Team: Resolved in the following code change.

M-42 | Not Updating Pairs Will Break LAV

Category	Severity	Location	Status
Protocol	Medium	LendingAssetVault.sol	Resolved

Description

When a deposit or withdraw happens in the `LendingAssetVault` all pairs should be updated because the `totalAssets` are increased unilaterally which affects the pairs.

There is a function which allows setting `_updateInterestOnVaults` to be set to false. If so, any updates to all pair at once will be skipped. This means users can manipulate pairs' utilization rates by depositing/withdrawing.

Recommendation

`_updateInterestOnVaults` is meant to be set to false if updating all vaults start causing OOG errors. Given the problem it creates and the facts that there is a limit to the maximum pairs that can be connected to a vault and that pairs can also be removed, the removal of `_updateInterestOnVault` is best.

Resolution

Peapods Team: Resolved in the following code change.

M-43 | Leverage Doesn't Work When FOT Is Enabled

Category	Severity	Location	Status
DoS	Medium	LeverageManager.sol	Acknowledged

Description PoC

Pods have a property `hasTransferTax` which when enabled, a fee-on-transfer is taken from the value to be transferred and the recipient receives less tokens.

This is a problem for the LeverageManager.addLeverage() function because it assumes the whole amount has been received and assigns that amount to `LeverageFlashProps.podAmount`.

Later, when the `podAmount` is requested from the `IndexUtils`, the transaction will revert because the `LeverageManager` contract doesn't have all the tokens.

Recommendation

Consider the fee on transfer aspect of the pod tokens when adding leverage.

Resolution

Peapods Team: Acknowledged.

M-44 | Swap Error Handling Causes DOS Of AutoCompounder

Category	Severity	Location	Status
Logic Error	Medium	AutoCompoundPodLp.sol	Resolved

Description PoC

In `_processRewardsToPodLp`, if a swap of the main reward token fails, an override feature kicks in to halve and store the next `_amountIn` to swap in `_tokenToPairedSwapAmountInOverride`.

A temporary DOS attack can be carried out as such:
1. Donate 50 wei of the main reward token (PEAS), assuming contract has no previous balance of PEAS.
2. Call `deposit` to trigger `_processRewardsToPodLp` where the small swap to Uniswap V3 would fail due to insufficient amountOut.
3. Half of 50 wei (i.e. 25 wei) will then be stored in the override mapping.
4. Contract will attempt to swap for another 6 times before the override amount is set to zero — preventing actual rewards from being processed.

Recommendation

Consider re-designing the error handling for the swap.

Resolution

Peapods Team: Resolved in the following code change.

M-45 | DOS Of depositFromPairedLpToken

Category	Severity	Location	Status
Logic Error	Medium	TokenRewards.sol: 128	Resolved

Description PoC

In `depositFromPairedLpToken`, if a swap of a reward token fails, an override feature kicks in to halve and store the next `_amountIn` to swap in `_rewardsSwapAmountInOverride`.

A temporary DOS attack is possible by making use of this feature:
1. Deposit 50 wei of the reward token. The small swap to Uniswap V3 would fail due to insufficient `amountOut`.
2. Half of 50 wei (i.e. 25 wei) will then be stored in `_rewardsSwapAmountInOverride`.
3. Contract will attempt to swap for another 6 times before the override amount is set to zero — preventing actual rewards from being processed.

Recommendation

Consider redesigning the override design for swap failures.

Resolution

Peapods Team: Resolved in the following code change.

L-01 | Wrong Address Assignment

Category	Severity	Location	Status
Logic Error	Low	Global	Resolved

Description

Numerous addresses are set as constant variables in the protocol. However, a multitude of these addresses are specific to Ethereum Mainnet and are either not deployed or occupied by an EOA on other chains, such as Arbitrum and Base.

This will lead to reverts when they are interacted with when calling `addLeverage()`, and make the functionality unusable outside of Ethereum Mainnet.

Here is a list of variables that are assigned a constant address that is either incorrect or not deployed outside of Mainnet:

- `DAI`
- `PROTOCOL_FEE_ROUTER`
- `REWARDS_WHITELIST`
- `STYETH`
- `YETH`,
- `WETH_YETH_POOL`,
- `V3_ROUTER`

Recommendation

Use an immutable instead of a constant for the addresses, and pass in the proper addresses in the constructor on deployment.

Resolution

Peapods Team: The issue was resolved in commit 9c1d3c2.

L-02 | Oracle Incompatible With Existing Pod Contracts

Category	Severity	Location	Status
Integration	Low	spTKNMinimalOracle.sol: 249	Acknowledged

Description

During `getPrices`, in order to correctly price each `pTKN` `_accountForCBRInPrice` is called internally. There it first checks unlocked = 1 which is the reentrancy guard in the pod contract. However, in older versions of the pod contract which are currently live, unlocked is not a uint but a boolean.

Therefore, this call to older pod contracts will always revert and fail, making the oracle incompatible with pods such as `pPEAS` and `pOHM` which hold the bulk of the protocol's TVL (see https://etherscan.io/token/0x027CE48B9b346728557e8D420Fe936A72BF9b1C7?a=0x80e9c48ec41af7a0ed6cf4f3ac979f3538021608#code).

Recommendation

Ensure that the oracle is compatible with the interfaces of older pod contracts.

Resolution

Peapods Team: Acknowledged.

L-03 | POD Ratio Can Be Manipulated

Category	Severity	Location	Status
Protocol	Low	WeightedIndex.sol	Resolved

Description PoC

Each underlying asset of the POD token has a weight assigned to it which determines how much of that token should be paid. If the weights for tokens A and B are 50 and 100, this means for each `token A`, 2 `token B` should be paid.

The `WeightedIndex` contract computes the `_tokenAmtSupplyRatioX96` variable by dividing the amount of underlying tokens the user is paying by the total amount of tokens held in the contract.

This variable is used in two places:
• to determine how much Pods will the user receive
• to calculate the amount of the other underlying tokens that the user must pay.
The problem is that `balanceOf` can be manipulated by anyone by sending tokens to the contract.
Let's take a look at the following example:
• Two tokens - A and B - with weights 50 and 100.
• Alice wraps `2A + 4B = 2 Pod`
• A third party sends `2A` to the contract. Total A = 4
• Bob comes and tries to wrap `2A + 4B`.
• Since the ratio is `2A / 4A = 1/2`, he receives 1 Pod and pays `2A + 2B`.

We can see how the ratio `A:B` changed from `1:2` to `1:1` which diverts from the expected ratio of the pod and the backing ratio users expect when entering a pod.

Recommendation

Consider tracking the underlying token balances in an internal mapping.

Resolution

Peapods Team: Resolved in the following code change.

L-04 | PodFlashSource Incompatible With Existing Pods

Category	Severity	Location	Status
Integration	Low	PodFlashSource.sol: 26	Acknowledged

Description

In the `paymentAmount` function, `FLASH_FEE_AMOUNT_DAI` is called on the pod contract to obtain the flash fee.

However, for existing pod contracts which are live (e.g. `pPEAS`) the flash fee is named `FLASH_FEE` instead. So calls to these pods will always fail. `LeverageManager.addLeverage` will therefore also fail if the flash source is a pod.

Recommendation

Ensure that `PodFlashSource` is compatible with both old and new pod contracts.

Resolution

Peapods Team: Acknowledged.

L-05 | Malicious Pod Could Allow Reentrancy

Category	Severity	Location	Status
Reentrancy	Low	LeverageManager.sol	Acknowledged

Description

If a malicious pod were to be used in `LeverageManager`, it would allow reentrancy in the add/remove leverage functions. The attacker would be able to access the critical `callback` function and provide arbitrary data to steal other users' funds.

This is currently prevented by owner-approved `flashSource` and `lendingPairs`. However, if a malicious pod were to be accidentally approved, the consequences would be severe.

Recommendation

Be extra careful about the approvals for `flashSource` and `lendingPairs`. Consider validating that the caller in `callback` is a whitelisted flash source.

Resolution

Peapods Team: Acknowledged.

L-06 | LendingAssetVault Is Not EIP-4626 Compliant

Category	Severity	Location	Status
ERC4626	Low	LendingAssetVault.sol	Resolved

Description

According to EIP-4626, `maxMint` should return 2**256 - 1 if there is no mint limit. The current implementation of the function returns `type(uint256).max - 1` which is equivalent to 2**256 - 2.

Recommendation

Return `type(uint256).max`.

Resolution

Peapods Team: Resolved in the following code change.

L-07 | Users Avoid Debonding Fee

Category	Severity	Location	Status
Logic Error	Low	WeightedIndex.sol: 234	Partially Resolved

Description

`debond()` checks if a user is withdrawing 98% or more of the total supply. If they are, then they do not have to pay a fee when debonding. A malicious user could take out a flashloan and bond to increase their share of the total supply to reach the target 98%, then debond right away to avoid paying fees.

Recommendation

Charge the fee to users unless they are debonding 100% of the total supply.

Resolution

Peapods Team: Resolved in the following code change.

L-08 | _clBaseFeed Should Be Set When BASE != USD

Category	Severity	Location	Status
Oracles	Low	spTKNMinimalOracle.sol	Resolved

Description

`clBaseFeed` is assigned to `CHAINLINK_BASE_PRICE_FEED` in the constructor. Later, in the Chainlink oracle when the QUOTE/BASE price is required, if this variable is not set the result will be QUOTE/USD.

Since there is no validation in the constructor, it's possible to not set the `clBaseFeed`. This will be okay for feeds where the BASE asset is USD, but otherwise the pricing will be incorrect.

Recommendation

The best solution is to add validation in the constructor.

Resolution

Peapods Team: Resolved.

L-09 | Dormant Token Rewards

Category	Severity	Location	Status
Logic Error	Low	DecentralizedIndex.sol: 447	Resolved

Description

When a flashloan is taken from `PodFlashSource.sol`, it calls `DecntralizedIndex::flash()`. The fee for the flashloan is taken in DAI and transferred to the `RewardsToken.sol` if DAI is the `PAIRED_LP_TOKEN` and not the reward token.

The DAI will remain dormant in the contract since it is not added to the `rewardsPerToken` mapping, and stakers will not receive the proper rewards during that time period.

Recommendation

Call `depositFromPairedLPToken()` after the fee is taken when DAI is the `PAIRED_LP_TOKEN` and not the reward token.

Resolution

Peapods Team: Resolved in the following code change.

L-10 | Unused Code In LAV

Category	Severity	Location	Status
Best Practices	Low	LendingAssetVault.sol	Resolved

Description

The `_assetDecimals` function in `LendingAssetVault` is not needed.

Recommendation

Consider removing the function.

Resolution

Peapods Team: Resolved.

L-11 | Lenders Can Avoid Socialization Of Bad Debt

Category	Severity	Location	Status
Logical Error	Low	LendingAssetVault.sol	Acknowledged

Description

During the liquidation of a borrower in a `FraxlendPair` vault, any debt that cannot be repaid (i.e. bad debt) is socialized among all lenders to the vault which includes the `LendingAssetVault` (LAV).

Within the LAV, the bad debt from one vault is further socialized among depositers to the LAV. The issue lies with lenders/depositors who can avoid the socialization of bad debt by front-running a `liquidate` call, therefore putting a greater burden on the other lenders.

Recommendation

Clearly document this risk to users.

Resolution

Peapods Team: Acknowledged.

L-12 | Removing Pairs In LAV Leaves Dirty State

Category	Severity	Location	Status
Logic Error	Low	LendingAssetVault.sol	Resolved

Description

When a pair is removed by calling `setVaultWhitelist(vault, false)`, the `vaultWhitelistAryIdx` and `vaultMaxPerc` variables for that pair are not cleared.

Recommendation

Clear these variables.

Resolution

Peapods Team: Resolved in the following code change.

L-13 | Whitelist Update Does Not Update All Vaults

Category	Severity	Location	Status
Logic Error	Low	LendingAssetVault.sol	Acknowledged

Description

The function `whitelistUpdate` can either update one specific vault or all vaults depending on the boolean passed in. When the boolean is false, all vaults but the calling vault is updated since `msg.sender` is passed as the `_vaultToExclude`.

It is unclear if this is the intended behavior as the natspec comments indicate that all vaults should be updated. However, if the calling vault was included in the update, `addInterest` may revert due to reentrancy protection in the `FraxlendPair` contract.

Recommendation

Be aware that not all vaults are updated when `whitelistUpdate(false)` is called, and consider updating the natspec comments for accuracy.

Resolution

Peapods Team: Acknowledged.

L-14 | Approved Parties Cannot removeLeverage

Category	Severity	Location	Status
Protocol	Low	LeverageManager.sol	Resolved

Description

Leverage can be added to positions by the owner of the position NFT or any approved party of that NFT. However, the opposite action - removing leverage - can be performed only by the owner of the NFT.

Recommendation

Document this behavior.

Resolution

Peapods Team: Resolved in the following code change.

L-15 | Stale Price Causes Division By 0

Category	Severity	Location	Status
Logic Error	Low	FraxlenPairCore.sol: 531	Acknowledged

Description

When a stale price is passed from the oracle, it will return the price as 0 and emit an alert in `_updateExchangeRate()`. If the `highExchangeRate` is 0, then it will end up reverting in the calculation of `_deviation` because it divides by 0.

This will lead to the event never being emitted and transactions will fail without a clear root cause.

Recommendation

Instead of emitting a log when there is bad price data, revert with a custom error to signify that the price is wrong.

Resolution

Peapods Team: Acknowledged.

L-16 | Oracle Reverts On Stale Price

Category	Severity	Location	Status
Oracles	Low	FraxlendPairCore.sol	Acknowledged

Description

When a stale price is passed from the oracle, it will return the price as 0 and emit an alert in `_updateExchangeRate()`. If the `highExchangeRate` is 0, then it will end up reverting in the calculation of `_deviation` because it divides by 0.

This will lead to the event never being emitted and transactions will fail without a clear root cause.

Recommendation

Instead of emitting a log when there is bad price data, revert with a custom error to signify that the price is wrong.

Resolution

Peapods Team: Acknowledged.

L-17 | Pod Flashloans Prevented Via Lock

Category	Severity	Location	Status
Logic Error	Low	spTKNMinimalOracle.sol: 249	Acknowledged

Description

`_accountForCBRInPrice()` validates that the pod is not currently locked, and otherwise reverts. When a user takes out a flashloan from a pod it locks the contract, and will prevent `getPrices()` from being called.

This will prevent `borrowAsset()` (and any other function that calls `updateExchangeRate()`) from executing from the pod that has taken the flashloan.

Recommendation

Without the lock validation, the oracle price would be manipulatable via a flashloan. Document that using `PodFlashSource` can lead to a revert from locking the pod.

Resolution

Peapods Team: Acknowledged.

L-18 | VotingPool Unstake Rounding

Category	Severity	Location	Status
Math	Low	VotingPool.sol	Acknowledged

Description

When users stake, the amount of voting tokens they get is determined by multiplying the deposited amount by the factor. In `unstake` the opposite is done - the unstaked amount is divided by the factor. This will result in users receiving less tokens than they initially deposited.

Recommendation

Document the discrepancy

Resolution

Peapods Team: Acknowledged.

L-19 | Possible Inflation Attack In AutoCompounder

Category	Severity	Location	Status
Logic Error	Low	AutoCompoundingPodLp.sol	Partially Resolved

Description

`AutoCompoundingPodLp` guards against the classic inflation attack by internally tracking deposits with the `_totalAssets` variable. However, the attack is still possible by sending reward tokens which are converted into assets before each action (e.g. deposit/withdraw). This increases `_totalAssets` without minting new shares.

If the `AutoCompoundingPodLp` is created by the factory contract, a minimum deposit is performed with 1e3 assets that creates 1e3 shares. This mitigates the effect of a donation but an attack is still possible.

Consider this example:
• After factory creation with min deposit: `_totalAssets = 1e3, totalSupply = 1e3`
• Attacker donates the equivalent of 1000e18 assets
• User deposits 1e18 assets:
• `_cbr: (1000e18 + 1e13) * 1e18 / 1e3` -> 1.00..03e36 (v large number)
• `convertToShares: 1e18 * 1e18 / 1.00..03e36` -> round down to 0
• User receives 0 shares and loses assets

The larger the donation, the higher the threshold will be for user's deposits to round down to 0 shares.

Recommendation

Consider performing a larger minimum deposit by the factory contract, which then increases the cost of the attack. Also, during deposit, after `convertToShares` is performed, checked that shares are not equal to 0 or else revert.

Resolution

Peapods Team: Resolved in the following code change.

L-20 | Approved Address Gets The Flash Loan Refund

Category	Severity	Location	Status
Logic Error	Low	LeverageManager.sol: 342	Resolved

Description

Removing leverage can only be done by position NFT owners, but adding leverage can be done by any approved user or operator. When adding leverage, the `msg.sender` provides their own funds as `pod` tokens, and `pod` token refunds are returned to the `msg.sender` as expected.

However, flash-loaned `pairedLpTokens` are also refunded to the `msg.sender`, not necessarily to the position owner. When the position owner has an over-collateralized position in Fraxlend, an approved user can flash loan more `pairedLpToken` than needed and receive the entire refund.

Recommendation

Reconsider who should receive the flash loan refunds. If the `msg.sender` is expected to receive them, rather than the position owner, consider documenting this behavior.

Resolution

Peapods Team: Resolved in the following code change.

L-21 | getFullUtilizationInterest() Calculation

Category	Severity	Location	Status
Logic Error	Low	VariableInterestRate.sol: 20	Acknowledged

Description

Inside of `getFullUtilizationInterest()`, the `_newFullUtilizationInterest` will always go down when the current utilization is less than `MIN_TARGET_UTIL`.

It does not take into consideration if the Fraxlend vault's utilization has actually gone up or down, and simply looks at if the utilization is within certain target ranges.

Recommendation

Be aware that this will occur, and that it will only go down based on your configuration of `MIN_TARGET_UTIL`.

Resolution

Peapods Team: Acknowledged.

L-22 | Unused Cached Value

Category	Severity	Location	Status
Logic Error	Low	FraxlendPairCore.sol: 242	Resolved

Description

`_exchangeRateInfo` is cached to memory in the `isSolvent` modifier of the `FraxlendPairCore` contract. However, this cached value is never used but the storage variable is being used instead.

Recommendation

Consider using the cached memory variable. Alternatively, remove the variable.

Resolution

Peapods Team: Resolved in the following code change.

L-23 | bondWeightedFromNative Fails For Multi-Asset Pod

Category	Severity	Location	Status
Logic Error	Low	IndexUtils.sol: 321	Acknowledged

Description

In `_swapNativeForTokensWeightedV2` , a loop is performed to swap WETH for each index token.

However by calling `swapExactTokensForTokensSupportingFeeOnTransferTokens`, all WETH (tokenIn) is swapped for tokenOut on the first loop iteration, leaving no remaining WETH for next iteration, if there are multiple assets in an index.

Then, there will be insufficient index tokens to bond when `_indexFund.bond` is later called.

Recommendation

Use the other Uniswap function `swapTokensForExactTokens` instead to get out a specific amount of each index token. Alternatively, calculate `getAmountsIn` to pass in exactly the amount of WETH needed before calling the swap function.

Resolution

Peapods Team: Acknowledged.

L-24 | Both buy And sell Fees Can Be Applied

Category	Severity	Location	Status
Logical Error	Low	DecentralizedIndex.sol	Resolved

Description

If a POD is transferred from a V2 pair, a buy fee is applied. If it's transferred to a V2 pair, a sell fee is applied. This means that if a POD is transferred from a V2 pair to the pair itself, both fees will be applied.

On top of that only the second fee is deducted from the amount to be subtracted. Imagine the following:
• Pair tries to transfer 20 tokens
• Buy fee is 50% so it pays 10 tokens
• Sell fee is 10% so it pays 2 tokens
• Since sell fee was the last recorded fee, the amount transferred to the recipient is (20 - 2) = 18
• Total transferred: 30 tokens

Recommendation

Instead of if-if, change the fee checks to if-else if

Resolution

Peapods Team: Resolved in the following code change.

L-25 | DOS Of Native Bond And Stake

Category	Severity	Location	Status
Logic Error	Low	IndexUtils.sol: 313	Resolved

Description

In `bondWeightedFromNative`, when `_stakeAsWell` is true, only half of msg.value is passed to `_swapNativeForTokensWeightedV2`. However, later during `_swapForIdxToken` the entire contract's balance of ETH is converted to WETH.

This results in a revert later on when _zapIndexTokensAndNative is called as _zap attempts to convert the remaining half of msg.value to WETH:

https://github.com/GuardianAudits/peapods-1/blob/main/contracts/Zapper.sol#L173.

Consequently, this protocol functionality becomes unusable.

Recommendation

If `_stakeAsWell` is true, only half of msg.value should be converted to WETH initially.

Resolution

Peapods Team: Resolved in the following code change.

L-26 | Unsuccessful Asp Reward Swaps

Category	Severity	Location	Status
Protocol	Low	LVF.sol	Acknowledged

Description

Flash loans from `UniswapV3` pools are taken for the paired token of the leveraged pod. If the second token in that pool matches with the reward token for the ASP, the swap will fail because swapping will be locked since flashloan is taken from that pool.

Recommendation

Make sure the Peapods team is aware of this

Resolution

Peapods Team: Acknowledged.

L-27 | Asp Rewards Impact removeLeverage

Category	Severity	Location	Status
Protocol	Low	LVF.sol	Resolved

Description

When leverage is being removed, the asp collateral has to be turned to `spTokens` via calling `redeem`. This action will trigger rewards distribution and will swap `pairedTokens` for pod tokens.

The price of the `pairedTokens` will drop while the price of the pod tokens will go up. When the `spToken` is unwrapped to LP token and liquidity is removed, because of the previous swap the user will receive more paired tokens and less pod tokens.

Recommendation

This should be resolved with implementing the one-sided liquidity formula.

Resolution

Peapods Team: Resolved in the following code change.

L-28 | LVF Bond Rounding

Category	Severity	Location	Status
Protocol	Low	LeverageManager.sol	Acknowledged

Description

When an `fToken` is bonded for a self lending pod, the `bond` function will transfer slightly less `fTokens` from the `LeverageManager` because of rounding. This will leave the `LeverageManager` with non-zero approval that will increase over time.

Also, if all of the paired assets are used up, the refund if statement won't be executed and the `msg.sender` won't receive these `fTokens`. They will be left in the contract and the next caller will have access to them.

Recommendation

Document this behavior

Resolution

Peapods Team: Acknowledged.

L-29 | Typos

Category	Severity	Location	Status
Logic Error	Low	Global	Resolved

Description

remvoe instead of remove is within the NatSpec.

Recommendation

Fix the typos

Resolution

Peapods Team: Resolved.

L-30 | Linear Interest Rate Will DOS Fraxlend

Category	Severity	Location	Status
DoS	Low	LinearInterestRate.sol	Acknowledged

Description

`LinearInterestRate` cannot be used because it implements the `IRateCalculator` interface where the `getNewRate` signature is `getNewRate(bytes,bytes)`. `FraxlendPairCore` uses `IRateCalculatorV2` with signature `getNewRate(uint256,uint256,uint64)`.

Recommendation

Change the `LinearInterestRate` to adhere to the `IRateCalculatorV2` interface.

Resolution

Peapods Team: Acknowledged.

L-31 | overrideBorrowAmt May Be Used Maliciously

Category	Severity	Location	Status
Logical Error	Low	LeverageManager.sol: 327	Acknowledged

Description

In `addLeverage` a user can intentionally borrow up to the solvency limit from Fraxlend by passing in desired `_overrideBorrowAmt`.

The caller of `addLeverage` will then receive any additional borrow tokens while putting the leveraged position on the edge of liquidation. This opens up an attack surface where borrowed funds leave the system.

This could also be abused by an approved account or if the user accidentally sets `isApprovedForAll` to his `positionNFT`.

Recommendation

Unless there are strong reasons to do so, do not allow a user to override borrow amount. Else, consider restricting the override borrow amount to provide some additional buffer from liquidation.

Resolution

Peapods Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits