BAIL security

ListaDAO
PCS V3 Integration

FINAL REPORT

August ‘2025

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	ListaDAO - PCS V3 Integration
Website	lista.org
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/lista-dao/lista-dao-contracts/tree/d603a5c208cef3d0f49ce6896300701f7c5b27fe
Resolution 1	https://github.com/lista-dao/lista-dao-contracts/tree/4be38d532df425aa333010360d4be4f84c40e783

2. Detection Overview

Due to the high amount of High/Medium issues as well as outstanding issues, we recommend a follow-up audit before deployment.

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High	11	5		2	2	2
Medium	13	5	2	2	1	3
Low	10	4	1	5		
Informational	4	1		3		
Governance						
Total	38	15	3	12	3	5

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

PancakeSwapV3LpProvider

The PancakeSwapV3LpProvider contract is the main contract that facilitates depositing PancakeSwap V3 positions as collateral into ListaDAO. It accepts V3 positions tokenized in the PancakeSwap NonFungiblePositionManager, as long as the position's token0 and token1 match what the provider expects. Each token0/token1 pair requires a separate provider contract to be deployed and integrated into the ListaDAO system.

When LP tokens are deposited, their USD value is computed using the PcsV3LpNumbersHelper, and an equivalent amount of LpUsd (a new token linked to each PancakeSwapV3LpProvider) is minted and deposited into the user's CDP to represent their collateral. The USD value of each LP token is periodically updated in the PancakeSwapV3LpProvider and synced downstream to the CDP, either when the user interacts with the provider again or through manual updates by the BOT role.

Upon deposit, LP tokens are transferred to the PancakeSwapV3LpStakingHub for reward farming.

Core Invariants:

INV 1: Each address can deposit up to maxLpPerUser LP tokens

INV 2: At the time of deposit, the value of the token being deposited must be at least minLpValue USD

INV 3: _getMaxCdpWithdrawable should return the maximum LpUsd that can be withdrawn from the user's CDP before it becomes liquidatable

INV 4: Only LP tokens relating to token0/token1 should be depositable

Privileged Functions

- syncUserLpValues
- batchSyncUserLpValues
- pause
- unpause

- setMaxLpPerUser
- setLpDiscountRate
- setMinLpValue

Issue_01	Syncing CDP position after transferring LP token allows users to re-enter and borrow at an inflated collateral amount
Severity	**High**
Description	_syncUserCdpPosition is now only called after transferring the lp token to the user *_withdrawAndSendRewards(tokenId, user);* *// sync user position* *_syncUserCdpPosition(user, false);* This is flawed as the transfer happens via safeTransferFrom which gives the execution control to the user. Since the CDP position has not yet been synced, a user can then borrow as if they haven't withdrawn the LP token
Recommendations	Invoke _syncUserCdpPosition before transferring the LP token but after invoking _removeToken
Comments / Resolution	

Issue_02	Users can re-enter after liquidation and borrow using collateral value that is yet to be cleared
Severity	**High**
Description	Remaining LP tokens are now transferred to the user after liquidation is finished. A user can steal value from these about to be removed LP tokens by re-entering via the onERC721Received function which will invoke _syncUserCdpPosition and hence mint lpUSD for these about to be removed LP tokens
Recommendations	Consider changing the code to not do the safeTransferFrom() call during the liquidation, and instead credit the user with their tokens in storage, and allow them to withdraw the tokens later on.
Comments / Resolution	

Issue_03	Wrong debt amount used in withdrawableAmount calculation
Severity	**High**
Description	The _getMaxCdpWithdrawable() function calculates the maximum collateral that can be safely withdrawn from a user's CDP. This calculation involves the current debt of the user, but the code incorrectly retrieves it by calling ICdp(cdp).vat().ilks(ilk). The art value from this call is actually the normalized debt for the entire collateral type, and not the normalized debt for the user's individual position. Because a user's individual debt will be less than the global debt, this causes the function to underestimate the withdrawable amount. As a result, if an LP token loses value, the system won't withdraw as much LpUsd as it should from the user's CDP (and likely wouldn't withdraw anything), effectively leaving users with free collateral for a devalued position.
Recommendations	Change the code to retrieve the user's debt using ICdp(cdp).vat().urns(ilk, user) and not ICdp(cdp).vat().ilks(ilk). Also, consider adding tests with multiple users on the same ilk to test the behavior when each user's debt differs from the global debt.
Comments / Resolution	Fixed. Now ICdp(cdp).vat().urns(ilk, user) is used to retrieve the user's debt

Issue_04	Oracle decimals is handled incorrectly
Severity	**High**
Description	There is inconsistency in the expected number of decimals for the oracle price. computeFairSqrtPriceX96 and the token calculation inside liquidation expects the oracle to return price in (8 + (18 - assetDecimals)) while the value calculation inside getLpValue expects the price to be in 8 decimals. When assetDecimals is not equal to 18, this will cause the calculations to be incorrect
Recommendations	Depending on how the oracle will be setup, correct the expected number of decimals
Comments / Resolution	Failed resolution. The decimal handling inside the getLPValue function is still incorrect because getAmounts returns token amount in 18 decimals while the getLPValue function expects the token amounts to be in asset's decimals. In more detail: - getAmounts() returns amount0/amount1 values with 18 decimal places (it explicitly does a normalization calculation to achieve this) - getTokenPrice() returns price0/price1 with 8 + (18 - token_decimals). The 8 is from the resilient oracle and the (18 - token_decimals) is from the normalization in the getTokenPrice() code - ORACLE_PRICE_DECIMALS is 8 decimal places So, for example, this means FullMath.mulDiv(amount0, price0, ORACLE_PRICE_DECIMALS) has decimals of: [18] + [8 + (18 - token_decimals)] - [8] = 18 + (18 - token_decimals) The actual decimals should just be 18 decimals always, so this is wrong anytime the token doesn't have 18 decimals

Issue_05	Partial liquidations can allow users to withdraw the underlying value
Severity	**High**
Description	As soon as a single liquidation is completed with leftover collateral, the liquidation record is cleared and remaining tokens are sent to the user. This is flawed as in case of partial liquidations there can be other ongoing liquidations of the same user or the user could have non-zero ink (collateral) in the cdp system which could now be unbacked. Partial liquidations can occur in the cdp system either due to total liquidation limit being hit or individual collateral liquidation limit being hit
Recommendations	One naive way is to keep the liquidation limits practically infinite so that partial liquidations never occur. Otherwise the challenge is to only release the collateral and clear liquidation record when it can be confirmed that there are no pending liquidations and 0 ink/debt for the user (can be achieved by making the user repay debt). For this the existing core contracts like clip/interaction will have to be modified since currently there is no way to know if a user has pending liquidations.
Comments / Resolution	Acknowledged.

Issue_06	_getMaxCdpWithdrawable() rounding can block withdrawals
Severity	High
Description	The _getMaxCdpWithdrawable() function calculates the maximum collateral that can be withdrawn from a user's CDP while keeping it healthy. This involves computing minRequiredCollateral from the current debt FullMath.mulDiv(art, rate, RAY), which is then used in the calculation FullMath.mulDiv(debt, mat, RAY). Notice that both of these steps round down. In contrast, the relevant check in the vat contract does not perform these divisions (except for computing spot) and simply verifies rate * art <= collateral * spot. Because _getMaxCdpWithdrawable() uses intermediate calculations that round down, it can underestimate the actual collateral required according to vat. This underestimation would cause the function to attempt to withdraw more collateral than is actually possible, which will revert. If this occurs, the system will fail to withdraw LpUsd when an LP token loses value.
Recommendations	Change the calculations in _getMaxCdpWithdrawable() to round up instead of down. Note that this may lead to a small amount of collateral not being withdrawn even if it can be, but this seems to be less of a concern compared to a blocked withdraw() function.
Comments / Resolution	Failed resolution. The calculation uses mat and doesn't consider the fact that the vat contract uses a rounded down spot (derived from price and mat) which can cause the required collateral value to be even higher

Issue_07	payByToken0AndToken1 doesn't handle scenario where totalWealthCanCoverAmount == false
Severity	**Medium**
Description	The payByToken0AndToken1 function assumes that amountLeft can always be covered by (token1Value +token0Value) and hence expects (amountLeft/token1MaxPayable) to <= 1 *if (amountLeft > 0 && amount1 > 0 && token1Value > 0) {* *uint256 token1MaxPayable = token1Value;* *uint256 token1AmountToSend = FullMath.mulDiv(amountLeft, amount1, token1MaxPayable);* *IERC20(token1).safeTransfer(recipient, token1AmountToSend);* *token1Sent = token1AmountToSend;* *amountLeft = 0;* But since this is no longer the case when totalWealthCanCoverAmount is false, (amountLeft/token1MaxPayable) will be greater than 1 and will cause token1AmountToSend to be greater than the available token balance
Recommendations	Handle shortage of token1Amount similar to how token0 is handled
Comments / Resolution	

Issue_08	Liquidators can end with in loss after a liquidation in case totalWealthCanCoverAmount == false
Severity	**Medium**
Description	The only loss protection mechanism available for liquidators is specifying the amount0Min and amount1Min (the minimum token amounts to be obtained after burning a LP token). But in case all the LP tokens have already been burned, this check will be skipped and when totalWealthCanCoverAmount is false, the liquidator can end up getting less value than amount which can even result in a loss for them *if (totalWealthCanCoverAmount) {* *require((token0Value + token1Value) >= amount, "PcsV3LpProvider: insufficient-lp-value");* *}*
Recommendations	Allow liquidators to specify a minimum to be received value and revert in case the value received is lower
Comments / Resolution	

Issue_09	Users can revert liquidations when liquidationEnded == true
Severity	**Medium**
Description	If the liquidation() function is called and the postLiquidation() function returns liquidationEnded == true, then the user's remaining LP tokens are returned to them. This is facilitated with a safeTransferFrom() call, which can give control flow to the user via the onERC721Received() hook. If a user intentionally sets up an onERC721Received() function that reverts, the entire liquidation transaction would revert. This means that a user always has the option to revert the final step of their liquidation. It's likely most users would not benefit from doing this, since they would be blocking their own LP tokens from being returned to them. However a dedicated attacker may be able to use this to inflict more monetary damage to the protocol than they lose themselves, and in general, users should not be able to block liquidations.
Recommendations	Consider changing the code to not do the safeTransferFrom() call during the liquidation, and instead credit the user with their tokens in storage, and allow them to withdraw the tokens later on.
Comments / Resolution	

Issue_10	Liquidation with isLeftOver as true might never happen
Severity	**Medium**
Description	When a user is liquidated, their collateral is auctioned, and each purchase during the auction eventually calls liquidation(). This function takes an isLeftOver boolean that indicates whether any collateral was returned to the user after the clip.take() call. The idea is that if collateral was returned, the auction is known to be finished and the user's userLiquidations storage is cleared so they can continue as normal. However, this does not consider the fact that an auction can end without returning any collateral to the user. This can happen if the auction lasts long enough for the user's entire LpUsd balance to be sold. In this case, isLeftOver will never be true, and the user would be stuck in a permanent liquidating state. This could result in some assets being frozen, because even if all LpUsd is sold, the user may still have LP positions or token0/token1 leftover. This is possible due to lpDiscountRate, which implies that the amount of LpUsd a user has does not match the actual value of their LP tokens.
Recommendations	Instead of using isLeftOver to determine when an auction has completed, refactor the logic to check clip.sales(auctionId) after the clip.take() call in AuctionProxy. If this check shows the auction is no longer active, make the call to the provider to clear the userLiquidations and related logic.
Comments / Resolution	Partially Resolved. Addressed 0 leftover scenario but the fix iterates through the entire active auctions list which is unbounded and can cause OOG/unprofitable liquidations due to high gas cost

Issue_11	Deposits can bypass the provide() function
Severity	**Medium**
Description	When a user deposits an LP token into the PancakeSwapV3LpProvider, the intended process is to approve the token to the provider and then call provide(). This function calls safeTransferFrom(), and the rest of the deposit logic runs in the provider's onERC721Received() function. However, there is currently nothing preventing a user from doing a safeTransferFrom() directly to the provider. This would also trigger the onERC721Received() function and complete the deposit as expected. The reason this is possible is because onERC721Received() does not check the operator address, which is the address that initiated the transfer. As a result, a user can bypass provide() and avoid the nonReentrant and whenNotPaused modifiers.
Recommendations	To enforce that the provide() function is used as expected, add a check in the onERC721Received() function that operator == address(this). UPDATE: A proper fix should also allow operator == pancakeStakingHub, since the staking hub also transfers LP tokens to the provider. So, the code could check operator == address(this) or operator == pancakeStakingHub.
Comments / Resolution	Failed resolution. The fix only adds whenNotPaused modifier to onERC721Received function rather than disallowing direct deposits. Hence although bypassing pause is now prevented direct deposits still offer more flexibility for the user (lack of non-reentrant modifier, allows an approved user to transfer on behalf of the owner) which is better to avoid and have already enabled a new high severity issue (see: Users can re-enter after liquidation and borrow using collateral value that is yet to be cleared)

Issue_12	dust increase can allow an attacker to create non-liquidateable positions and extract value
Severity	**Medium**
Description	The underlying cdp system has a dust (minimum debt) parameter which may be adjusted. In case a position has debt below dust, its collateral amount cannot be modified unless either the debt is fully cleared or the debt is increased above dust. An attacker can leverage this to make syncs between lpToken value and the collateral amount to fail. This allows them to bypass liquidations in case of falling LP token value and possibly eventually borrow a higher amount than collateral backing Eg: 1. dust is being increased from 100 to 200 2. attacker deposits lpToken currently worth 1M usd, gets 800k worth of collateral and takes a debt of 100 3. after the dust increase, the lpToken value drops to 400k. but _syncUserCdpPosition cannot be invoked to reduce the collateral because frob call will revert since debt is below dust 4. user now takes 600k worth of debt and will be in profit
Recommendations	One naive way to fix this is to make sure that the dust value is never increased. Else a new function can be created that will exchange the collateral for paying back the dust amount of debt
Comments / Resolution	Acknowledged. The client never plans to increase dust

Issue_13	Liquidations will revert in case the token is delisted
Severity	**Medium**
Description	When a collateral is delisted, liquidations are meant to successfully happen but deposits will be reverted. But since _syncUserCdpPosition (invoked inside liquidation) always attempts to mint and deposit the remaining value the liquidations will revert until leftover amount becomes 0
Recommendations	Only attempt to deposit in case the token is not blacklisted
Comments / Resolution	Fixed. Now the liquidation function sends remaining LP tokens to the user rather than minting new ink.

Issue_14	_syncUserCdpPosition() withdrawing logic is fragile
Severity	**Medium**
Description	The _syncUserCdpPosition() function updates a user's LpUsd collateral to reflect the current value of their LP positions. However, it cannot always withdraw the full amount it intends because it can't reduce collateral to the point where the CDP becomes liquidatable. This creates fragile behavior. If a user's LP token value drops sharply, the function can only pull LpUsd down to the liquidation edge, not the full loss. Users may get a short window to react to avoid liquidation, and benefit from the collateral that was not fully reduced.
Recommendations	A full fix for this concern likely requires a larger refactor. One potential idea is to somehow mark the user as "ready to liquidate" when _syncUserCdpPosition() cannot withdraw the full desired LpUsd.
Comments / Resolution	Acknowledged.

Issue_15	Collateral cannot be recovered in case of leftover dust bad debt
Severity	**Medium**
Description	In case the debt in auction is <= chost, it has to be repaid in full (or the entire collateral has to be bought). Hence the minimum amount of collateral required for this would be min(chost/topPrice, totalLot). Unlike normal collaterals here the actual underlying collateral can be worth less than this minimum amount in case the LP token value drops sharply. If this happens, the auction can never be completed and this collateral cannot be recovered. Note that the yank function available in clip wouldn't help here to recover the collateral (unlike normal scenarios) since lpUSD is only a place holder collateral
Recommendations	Addressing this issue completely would require a large refactor. One partial mitigation is to set minLpValue sufficiently high so that it would require a very large value drop to create a situation where an LP token isn't worth enough to cover the minimum collateral purchase.
Comments / Resolution	Partially Resolved. Now the execution doesn't revert in case the collateral value obtainable is less than amount and hence the team can liquidate suffering a loss and still obtain the leftover collateral (although in case there are multiple LP tokens, the protocol can only obtain assets from one. Further edge cases of non-bad dust debt but amount not retrievable by burning a single LP token also remain). The team plans to set a sufficiently high minLpValue and low LTV as safeguards against this scenario from ever occurring.

Issue_16	Transferring leftover tokens to the owner can cause reverts if the owner is blacklisted
Severity	**Low**
Description	When there are leftover tokens after a liquidation, the tokens are sent out to the owner. This can cause a revert in case the token is a blacklistable one and the owner was blacklisted. This usually harms the owner (ie. if an owner was blacklisted in one token, they can lose a lot more value) since the price will continue to drop till the leftover amount will be 0. But if the auction will reset before the required price drop, it can turn into an opportunity to farm redo auction incentives Eg: lot = 100 debt = 50 if cusp == 60% (hence allowed drop == 40%), then leftover amount == 0 will only happen after a price drop of 50%, before which the auction will be reset and the incentive paid out making it farmable
Recommendations	If blacklistable tokens will be used, it is better to use a pull mechanism to distribute funds ie. internal accounting + recipient retrieves funds from the contract
Comments / Resolution	Acknowledged.

Issue_17	maxLpPerUser check performed is off by 1
Severity	Low
Description	The length check allows users to deposit tokens even when the current length is maxLpPerUser. After the deposit, the user will now have maxLpPerUser + 1 number of lpTokens in total *function _deposit(address user, uint256 tokenId) internal {* *// check if user has reached the max LP limit* *require(userLps[user].length <= maxLpPerUser,* *"PcsV3LpProvider: max-lp-reached");*
Recommendations	Change the check to require(userLps[user].length < maxLpPerUser, "PcsV3LpProvider: max-lp-reached");
Comments / Resolution	Fixed as per the recommendation.

Issue_18	userTotalLpValue is not decreased after lpToken burning
Severity	Low
Description	userTotalLpValue is supposed to be the sum of the lpToken values of an user. But during liquidation when a lpToken is burned its individual value is not decreased from this sum causing the value to be incorrect
Recommendations	Decrease the value of the individual position from userTotalLpValue
Comments / Resolution	Fixed. Now _syncUserLpTotalValue is invoked inside _burnLp.

Issue_19	Liquidators cannot enforce the actual token0/token1 minimum returned to them
Severity	**Low**
Description	When a liquidator calls buyFromAuction() for a PancakeSwapV3LpProvider liquidation, they pass amount0Min, amount1Min, and tokenId. The amount0Min and amount1Min checks only apply if the LP token is actually burned, and they only constrain the tokens returned from that burn. They do not constrain the total token0/token1 the liquidator ultimately receives from the liquidation flow. As a result, liquidators cannot express the true minimum amounts of token0/token1 they want to receive from the entire liquidation. If conditions change before their transaction executes, they may receive more or less of each token than expected.
Recommendations	Consider allowing the user to specify the minimum token0/token1 amount returned to them from the liquidation, not just the burn step.
Comments / Resolution	Acknowledged.

Issue_20	PancakeSwapV3LpProvider does not enforce a specific fee tier
Severity	Low
Description	Each PancakeSwapV3LpProvider is linked to a specific token0/token1 pair, and deposited LP positions must be from pools with that token pair. However, nothing enforces that the position comes from a pool with a specific fee tier. Since PancakeSwapV3 distinguishes pools by both tokens and fee, positions from different pools can be deposited into the same provider.
Recommendations	Consider whether this behavior is expected. If it is only intended for LP positions to be from one pool, then consider restricting to a specific fee in the _verifyLp() function.
Comments / Resolution	Acknowledged.

Issue_21	LpUsd is not burned when isLeftOver is false
Severity	Low
Description	When the liquidation() function is called with isLeftOver == false, the AuctionProxy sends the purchased LpUsd to the provider contract. However, the current implementation of the isLeftOver == false case doesn't burn these tokens, so they remain in the PancakeSwapV3LpProvider and accumulate over time.
Recommendations	In the isLeftOver == false case, add logic to burn the LpUsd that the provider receives from AuctionProxy.
Comments / Resolution	Fixed. Now the lpUSD is burned inside liquidation.

Issue_22	Incorrect error description
Severity	**Informational**
Description	The onlyStakingVault modifier has PancakeSwapStakingHub: only-staking-vault as the error description instead of PcsV3LpProvider: caller-only-staking-vault
Recommendations	Change the error description to PcsV3LpProvider: caller-only-staking-vault
Comments / Resolution	Acknowledged. The error changed “only-staking-vault” -> “caller-only-staking-vault” but didn’t change “PancakeSwapStakingHub” -> “PcsV3LpProvider”.

Issue_23	decimals is incorrectly mentioned as 8 instead of 18 in several places
Severity	**Informational**
Description	Throughout the codebase asset values are eventually scaled to 18 decimals but return value description of several functions incorrectly mention 8 decimals *@return appraisedValue the appraised value in USD with 8 decimal places*
Recommendations	Change the error description to 18 decimals
Comments / Resolution	Acknowledged. The description is still incorrect in multiple places: https://github.com/lista-dao/lista-dao-contracts/blob/4be38d532df425aa333010360d4be4f84c40e783/contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol#L784-L786 (the USD value should be 18 decimal places) https://github.com/lista-dao/lista-dao-contracts/blob/4be38d532df425aa333010360d4be4f84c40e783/contracts/ceros/provider/pancakeswapLpProvider/libraries/PcsV3LpNumbersHelper.sol#L177-L179 (the USD value should be 18 decimal places) https://github.com/lista-dao/lista-dao-contracts/blob/4be38d532df425aa333010360d4be4f84c40e783/contracts/ceros/provider/pancakeswapLpProvider/libraries/PcsV3LpNumbersHelper.sol#L134-L143 (the price returned from getTokenPrice() has decimals of 8 + (18 - token_decimals)) https://github.com/lista-dao/lista-dao-contracts/blob/4be38d532df425aa333010360d4be4f84c40e783/contracts/ceros/provider/pancakeswapLpProvider/libraries/PcsV3LpNumbersHelper.sol#L86 (the price returned from getTokenPrice() has decimals of 8 + (18 - token_decimals))

Issue_24	lpValue will always be 0 in WithdrawLp event
Severity	Low
Description	The WithdrawLp event emits lpValues[tokenId] which will always be 0 since the storage is cleared inside the _removeToken function
Recommendations	Cache the value before clearing the storage and emit this
Comments / Resolution	Acknowledged. The value emitted is the old value and not the current value. If the current value has to be emitted, it should be cached after invoking _syncUserCdpPosition.

Issue_25	The _removeToken function consumes excessive gas for large user LP lists.
Severity	Low
Description	The _removeToken function iterates over all userLps to remove a specific token. If the list of userLps is large and the target token is near the end, the gas cost may become high. Currently, the check that the list is smaller than maxLpPerUser helps prevent _syncUserLpTotalValue from running out of gas.
Recommendations	Consider using an EnumerableSet, which enables adding and removing tokenIds in O(1) time complexity, instead of the current O(n) approach.
Comments / Resolution	Acknowledged.

PancakeSwapV3LpStakingHub

The PancakeSwapV3LpStakingHub contract is the middle man between PancakeSwapV3LpProviders and MasterChefV3. It handles all the interactions (this includes depositing, withdrawing, harvesting CAKE rewards and burning) with MasterChefV3. An emergencyMode is also introduced which can be triggered by the owner when MasterChefV3 is in emergency. In this mode all the tokens are withdrawn from MasterChefV3 and remain in the staking hub contract until MasterChefV3 comes out of emergency. This contract is shared by all the PancakeSwapV3LpProviders

Core Invariants:

INV 1: All LP tokens in the system should be held in the PancakeSwapV3LpStakingHub

INV 2: Each LP token held in the contract should exist in the tokenIds array

Privileged Functions

- transferOwnership
- renounceOwnership
- emergencyWithdraw
- stopEmergencyMode
- registerProvider
- deregisterProvider

Issue_26	harvest function is not invoked before burning causing liquidations to revert
Severity	**High**
Description	The _burnAndCollectTokens function attempts to burn the lpToken from masterChefV3 without harvesting its rewards. This is flawed as the burn call will revert due to non-zero reward which will upstream cause the liquidations to revert as well
Recommendations	Invoke masterChefV3.harvest before calling burn
Comments / Resolution	Fixed. Now harvest is invoked before burn in case pendingCake is > 0

Issue_27	The fee on rewards can be bypassed if the pair contains the reward as one of its token
Severity	**Medium**
Description	If _burnAndCollectTokens correctly collects the rewards as intended (i.e., during the decreaseLiquidity call), those rewards are included in the position's tokens through the before-and-after balance calculation. Subsequently, the tokens are transferred directly to the provider. As a result, the before-and-after balance calculation for the rewards yields zero, and no fee will be deducted in the provider contract.
Recommendations	Consider harvesting the rewards *after* the before-after balance calculation.
Comments / Resolution	Fixed. Now reward is harvested before performing the before-and-after balance calculation to determine the positions's token amounts

Issue_28	Already accrued rewards are not handled in case of emergencyWithdraw
Severity	**Medium**
Description	In emergency mode all the tokens are withdrawn from masterchef. This will send the already accrued rewards to the PancakeSwapV3LpStakingHub contract. But these rewards are not handled in any way and will remain in the contract forever unclaimed
Recommendations	If rewards have to be distributed to the users themselves, consider adding an additional function inside PancakeSwapV3LpStakingVault that allows the owner to harvest rewards for any user. This can then be invoked before calling the emergencyWithdraw function. Or add a sweep function so that the rewards are not lost
Comments / Resolution	Fixed. The emergencyWithdraw function has been removed and withdrawals happen as if normally even when MasterChef is in emergencyMode. Note that the emergencyMode boolean is still defined but no longer used in the code.

Issue_29	emergencyMode doesn't integrate with core Provider functionalities properly causing DOS
Severity	**Medium**
Description	When the underlying masterchef goes into emergency, the owner can trigger emergencyMode in the PancakeSwapV3LpStakingHub contract. In this mode all the tokens are withdrawn from the masterchef contract to the PancakeSwapV3LpStakingHub contract. But the current codebase doesn't handle token deposits, withdrawals and liquidation/burning correctly 1. deposits: Users can still deposit new LP tokens during emergencyMode via the provider. This newly deposited token will now be present in masterchef rather than the stakingHub contract. Since the stakingHub contract attempts to deposit all the tokenIds to the masterChef inside stopEmergencyMode, this will revert and the contract can never come out of emergencyMode causing a permanent DOS 2. Withdrawals and liquidation: Both withdrawals and liquidation will attempt to either withdraw or burn the tokenId from masterChef contract. Since the token is no longer present in the masterChef contract, these calls will revert
Recommendations	Always check the emergencyMode flag before attempting to deposit into or withdrawing/burning from the masterchef contract. In case the flag is on, expect the token to be/have to be present in the PancakeSwapV3LpStakingHub contract
Comments / Resolution	Fixed. Now tokens are only deposited to masterChef if it is not in emergency and withdrawals/burning only attempts to withdraw/burn the tokenId from masterChef in case it was staked earlier.

Issue_30	Unbounded tokenIds array iteration
Severity	**Medium**
Description	In the PancakeSwapV3LpStakingHub contract, the _removeTokenRecord(), emergencyWithdraw(), and stopEmergencyMode() functions may iterate over the entire tokenIds array. Although each user can only deposit up to maxLpPerUser LP tokens, there can be arbitrarily many users, so this array could grow to be arbitrarily large. This means it is technically possible for these functions to become uncallable if iterating through the full array requires more gas than the block gas limit. An attacker might be incentivized to intentionally exploit this with the _removeTokenRecord() function. If the array becomes large enough, tokens near the end would not be reachable within the block gas limit, which would make their liquidation impossible. A rough estimate suggests this limit could be reached with somewhere between a few thousand and several tens of thousands of entries, given typical block gas limits are in the tens of millions and since SLOAD costs 2,100 gas. Since each LP token must be worth at least minLpValue USD at deposit time, setting minLpValue sufficiently high would likely make such an attack economically infeasible.
Recommendations	Consider refactoring the code to not require iterating through the entire tokenIds array, especially for the _removeTokenRecord() function. Alternatively, consider imposing a cap on the length of the tokenIds array to prevent it from growing large enough to block function execution. Finally, ensure that minLpValue is set high enough to make such an attack prohibitively expensive.
Comments / Resolution	Fixed. tokenIds array is now deprecated and no longer used. Note that the tokenIds variable is still defined even though it is not used.

Issue_31	Liquidation availability when paused is inconsistent
Severity	**Low**
Description	Even though the liquidation function of PancakeSwapV3LpProvider doesn't have whenNotPaused modifier the function will revert incase PancakeSwapV3LpStakingHub contract is paused as burnAndCollect has the whenNotPaused modifier. Also note that the liquidation/daoBurn availability is inconsistent when comparing different type of providers as well (eg: mbtcProvider doesn't allow daoBurn when paused but not so in HelioETHProvider and PancakeSwapV3LpProvider)
Recommendations	Maintain consistency for the liquidation function availability
Comments / Resolution	Partially Resolved. whenNotPaused modifier is now explicitly added to the liquidation function. But the inconsistency b/w different providers still remain

PancakeSwapV3LpStakingVault

The PancakeSwapV3LpStakingVault contract handles reward harvesting and fee collection. Users wanting to collect CAKE rewards from their deposited PCSV3LpToken should invoke the batchClaimRewards function of this contract which then invokes vaultClaimStakingReward function of the specific PancakeSwapV3LpProvider contract. A portion of the reward is collected as fees and the fee percentage is custom for each provider. If the PCSV3LpToken is being withdrawn directly from the provider contract, the fee is still captured by this contract through the feeCut function

Core Invariants:

INV 1: Reward claiming operations (outside of minting/burning flow) must be initiated by calling the PancakeSwapV3LpStakingVault

INV 2: All harvested rewards must have a fee applied, based on the LP token's provider contract

Privileged Functions

- collectFees
- setLpProxy
- setLpProviderFeeRate
- registerLpProvider
- deregisterLpProvider
- pause
- unpause

Issue_32	batchClaimRewardsWithProxy() is not currently callable
Severity	Informational
Description	The batchClaimRewardsWithProxy() function is callable by the LpProxy contract, which is out of scope for this review. However, it can be seen that the current LpProxy implementation cannot call batchClaimRewardsWithProxy() on the PancakeSwapV3LpStakingVault because it does not support passing the required argument types. As a result, batchClaimRewardsWithProxy() is currently unusable.
Recommendations	Consider updating the LpProxy in a future upgrade to the codebase.
Comments / Resolution	Acknowledged.

Issue_33	_batchClaimRewards() does not validate providers array
Severity	High
Description	The _batchClaimRewards() function does not enforce that the elements of the providers array are registered in the lpProviders storage. This can allow for stealing any reward tokens that exist in the contract by using a malicious provider contract that implements vaultClaimStakingReward() to return an arbitrary amount to steal. This can be used to steal the availableFees specifically, since the availableFees are reward tokens that are held by the contract.
Recommendations	To prevent theft of reward tokens, add a check that each element of the providers input array exists in the lpProviders storage.
Comments / Resolution	Acknowledged. No changes as of resolution commit 4be38d532df425aa333010360d4be4f84c40e783.

PcsV3LpNumbersHelper

The PcsV3LpNumbersHelper is a library used by the PancakeSwapV3LpProvider contract to obtain the fair underlying amounts of a LP token. The concept of fair amounts is used in order to be resilient against underlying pool manipulations. It is inspired by Maker's GUniLpOracle. The amounts returned are always scaled to 18 decimals

Core Invariants:

INV 1: The values returned from getAmounts don't depend on the pool state, and only depend on the position and the oracle price of each token

Privileged Functions

- none

Issue_34	Using unchecked can save gas and maintain perfect match with original implementation
Severity	**Informational**
Description	The original implementation uses solidity version 0.6.12 which is similar to unchecked by default. Hence performing the calculations inside an unchecked block will maintain perfect match with the original implementation and also save on gas
Recommendations	Wrap the calculations inside an unchecked block
Comments / Resolution	Fixed, however note that this suggestion does not apply to the getAmounts() function and its unchecked block can be removed.

PcsV3LpLiquidationHelper

The PcsV3LpLiquidationHelper is a library used by the PancakeSwapV3LpProvider during liquidations. It has two functions. The payByToken0AndToken1() function handles repaying the liquidator with the token0 and token1 amounts from burned LP tokens. The sweepLeftoverLpUsd() function removes any remaining LpUsd tokens after a liquidation auction completes.

INV 1: Liquidators are always paid with token0 first, and token1 is used only if token0 alone is insufficient

Privileged Functions

- none

Issue_35	sweepLeftoverLpUsd() attempts to burn from wrong location
Severity	**Low**
Description	The sweepLeftoverLpUsd() function attempts to burn any free LpUsd that was returned to the user after the liquidation auction finished. However, this is incorrect based on the logic in the AuctionProxy. In the AuctionProxy code, the user's leftover LpUsd is transferred from them to the Interaction contract, which then holds it as an actual LpUsd balance. Because of this, sweepLeftoverLpUsd() will never burn any tokens, since they are in the Interaction contract and not in the user's balance.
Recommendations	Change the code to burn the LpUsd from the Interaction contract. Alternatively, change the AuctionProxy code to not move the user's free LpUsd balance to the Interaction contract.
Comments / Resolution	Fixed. Now lpUSD is burned inside the liquidation function.

LpUsd

The LpUsd contract is an ERC20 token that represents the liquidity in Usd for a specific collateral via a provider. It inherits Openzeppelin's ERC20 contract and adds mint and burn functionalities that are both guarded by onlyMinter modifier. The minter is usually the corresponding provider contract. A setMinter function is also added which allows the owner to update the minter

Core Invariants:

INV 1: The sum of all user's balance equals the total supply of the token

INV 2: Only the minter contract can call mint and burn

Privileged Functions

- transferOwnership
- renounceOwnership
- setMinter
- mint
- burn

No issues found.

Interaction

The Interaction contract is the primary entry point for users and is the only way they can interact with the vat, jar, join, and related components. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope.

The changes in scope for this review include the removal of the setDutyCalculator() and getNextDuty() functions, and the addition of logic allowing a liquidator to pass bytes data in a liquidation call.

Issue_36	Users can block calls to withdraw() and buyFromAuction()
Severity	**High**
Description	The Interaction contract transfers tokens from a user's balances and operates on their urn, which requires authorization in the vat contract. This is set in the deposit() function, where vat.behalf() is called to authorize the Interaction contract on behalf of the user. Although this authorization is established during deposit(), the user can revoke it at any time by calling nope() directly on the vat contract. If revoked, the Interaction contract cannot operate on the user's behalf. This would block functions like withdraw(), which the PancakeSwapV3LpProvider relies on to remove LpUsd collateral when the LP token loses value. It would also block the final step of buyFromAuction(), which transfers leftover collateral from the user at the end of a liquidation. This means that a user can intentionally disrupt the final stage of the liquidation process. Note that this issue is outside the defined scope of the review but was identified during the audit and communicated to the team.
Recommendations	In any Interaction contract function that operates on behalf of the user, ensure a call to vat.behalf() is included within that function rather than relying solely on the call in deposit(). Additional functions that would require this call include withdraw(), buyFromAuction(), and borrow().
Comments / Resolution	Fixed as recommended. This change does not alter the user's ability to call vat.nope(). The only difference is the addition of extra vat.behalf() calls in the core functions of the Interaction contract. This ensures that even if the Interaction contract is the target of a vat.nope(), the core functionality remains unaffected, since vat.behalf() will re-establish the required authority for it to operate on the user's vat.

AuctionProxy

The AuctionProxy library is used by the Interaction contract during liquidation calls. It contains functions for interacting with the dog and clip components during liquidations, and also integrates with each provider (including the PancakeSwapV3LpProvider) if configured. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope.

The changes in scope for this review include refactoring existing logic into the _payMaxHay() helper function, and adding logic to allow a liquidator to pass bytes data to provider contracts during liquidation.

Issue_37	Leftover tokens are sent incorrectly to the liquidator specified address rather than the position owner
Severity	**High**
Description	In case leftover amount is non-zero, AuctionProxy invokes the liquidation() function of BaseTokenProvider(this is applicable to some other providers as well) with _recipient set to param.receiverAddress rather than urn. This will cause the leftover tokens to be sent to the param.receiverAddress rather than the position owner/urn
Recommendations	Either pass in urn as the _recipient or modify BaseTokenProvider(and other providers) to send assets to the user in case _isLeftover is true
Comments / Resolution	Fixed. Now recipient address is set to urn

BaseTokenProvider

The BaseTokenProvider contract is an abstract contract that is meant to be inherited by specific collateral provider contracts. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope

The changes in scope for this review include adding a new liquidation() function to conform to the new liquidation interface.

No issues found.

PumpBTCProvider

The PumpBTCProvider contract is the provider contract for PumpBTC collateral. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope

The changes in scope for this review include refactoring the original liquidation() function to be public, and introducing a wrapper function that conforms to the new liquidation interface.

No issues found.

mBTCProvider

The mBTCProvider contract is the provider contract for mBTC collateral. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope

The changes in scope for this review include refactoring the original liquidation() function to be public, and introducing a wrapper function that conforms to the new liquidation interface.

No issues found.

HelioETHProvider

The HelioETHProvider contract is the provider contract for ETH collateral. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope

The changes in scope for this review include adding a new liquidation() function to conform to the new liquidation interface.

Issue_38	Incorrect liquidation function parameters
Severity	**High**
Description	The new liquidation function in HelioETHProvider is declared as liquidation(address,address,uint256,bool). This is incompatible with the AuctionProxy, which now calls liquidation(address,address,uint256,bytes,bool). Because the bytes parameter is missing, every liquidation attempt involving the HelioETHProvider will revert.
Recommendations	Change the liquidation() function to take the correct number and types of parameters to match liquidation(address,address,uint256,bytes,bool). Ensure this is thoroughly tested before upgrading.
Comments / Resolution	Fixed as recommended.

HelioProviderV2

The HelioProviderV2 contract is the provider contract for BNB collateral. Only minor changes were made to this contract within the audit scope. This part of the review was performed as a differential audit, and any issues present in the original implementation outside the modified code are not in scope

The changes in scope for this review include refactoring the original liquidation() to conform to the new liquidation interface.

No issues found.