Lista Dao - Audit 2

Security Assessment

CertiK Assessed on Sept 4th, 2025

CertiK Assessed on Sept 4th, 2025

Lista Dao - Audit 2

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Binance Smart Chain (BSC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 09/04/2025 Final report published on 09/04/2025

Vulnerability Summary

28	17	1	0	10	0
Total Findings	Resolved	Multi-Sig & Timelock	Partially Resolved	Acknowledged	Declined

1	Centralization	1 Multi-Sig & Timelock	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
4	Major	4 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
7	Medium	4 Resolved, 3 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
12	Minor	9 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	4 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | LISTA DAO - AUDIT 2

CODEBASE | LISTA DAO - AUDIT 2

Repository

pre

Commit

d603a5c208cef3d0f49ce6896300701f7c5b27fe

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | LISTA DAO - AUDIT 2

This audit was conducted for Lista Dao to evaluate the security and correctness of the smart contracts associated with the Lista Dao - Audit 2 project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Static Analysis, Formal Verification, and Manual Review.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

FINDINGS | LISTA DAO - AUDIT 2

28	0	1	4	7	12	4
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Lista Dao to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 28 issues were identified. Leveraging a combination of Static Analysis, Formal Verification & Manual Review the following findings were uncovered:

ID	Title	Category	Severity	Status
LDA-24	**Centralization Related Risks**	**Centralization**	**Centralization**	**3/6 Multi-Sig, 24h Timelock**
LDA-01	Incorrect Debt In `_getMaxCdpWithdrawable()` Prevents LP Release	Logical Issue	Major	Resolved
LDA-02	Emergency Withdrawal Rewards Are Locked	Logical Issue	Major	Resolved
LDA-03	Unharvested Rewards Disrupt Liquidation Flow	Logical Issue	Major	Resolved
LDA-17	Leftover Tokens Transferred To Receiver Instead Of User During Liquidation	Volatile Code	Major	Resolved
LDA-04	No Cap On Fees	Logical Issue	Medium	Acknowledged
LDA-05	Misuse Of `nonReentrant`	Coding Issue	Medium	Resolved
LDA-06	Uninitialized State Variable `dutyCalculator`	Coding Issue	Medium	Resolved
LDA-07	Unbounded Loop In `batchClaimRewards` May Lead To Out-Of-Gas Exception	Logical Issue	Medium	Acknowledged
LDA-08	Stale LP Oracle Value	Inconsistency	Medium	Acknowledged

ID	Title	Category	Severity	Status
LDA-09	Users Can Still Deposit When Paused	Logical Issue	Medium	Resolved
LDA-10	Deprecated SafeApprove Pattern May Interrupt The Reward Payout	Logical Issue	Medium	Resolved
LDA-11	Block Timestamp Used As Deadline Parameter In Uniswap AMM	Logical Issue	Minor	Resolved
LDA-12	Emergency Mode Withdrawal Failure	Logical Issue	Minor	Resolved
LDA-13	Deregistration Operation May Lock Funds	Logical Issue	Minor	Resolved
LDA-14	LpUsd Not Handled During Liquidation	Inconsistency	Minor	Resolved
LDA-18	Missing User Position Sync After Complete Liquidation	Volatile Code	Minor	Acknowledged
LDA-19	Liquidation Order Not Enforced By LP Value	Inconsistency	Minor	Acknowledged
LDA-20	Missing Ongoing Status Check In `Liquidation` Function	Volatile Code	Minor	Resolved
LDA-21	Fixed Discount In Price Feed May Cause Mispricing	Volatile Code	Minor	Resolved
LDA-23	Unsettled Liquidation Status	Inconsistency	Minor	Resolved
LDA-25	Approved Address Can Approve Other Addresses	Design Issue	Minor	Resolved
LDA-26	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
LDA-28	Pyth Unchecked PublishTime	Coding Issue	Minor	Resolved

ID	Title	Category	Severity	Status
LDA-15	Unused Function Parameters Can Lead To False Assumptions	Logical Issue	Informational	Acknowledged
LDA-16	LP Price Sync In `PancakeSwapV3LpProvider::_deposit()`	Inconsistency	Informational	Acknowledged
LDA-22	Price Oracle Decimal Assumes A Fixed Value	Inconsistency	Informational	Acknowledged
LDA-29	Unchecked Arithmetic In Token Amount Normalization May Cause Overflow	Volatile Code	Informational	Acknowledged

LDA-24 | Centralization Related Risks

Category	Severity	Location	Status
Centralization	**Centralization**		**3/6 Multi-Sig, 24h Timelock**

Description

In the contract **DynamicDutyCalculator**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function setCollateralParams(): Set parameters for a collateral token.
- function setPriceRange():Set the price range for the dynamic interest rate mechanism's effect.
- function setDutyRange():Set the max duty and min duty for the case that price goes outside of range.
- function setDelta():Set the minimum price change required to update duty.
- function file():Set the address of the specified contract.

In the contract **DynamicDutyCalculator**, the role **INTERACTION** has authority over the following functions:

- function calculateDuty(): Calculate the duty for a collateral token based on the lisUSD price.

In the contract **PancakeSwapV3LpProvider**, the role **MANAGER** has authority over the following functions:

- function unpause(): resume contract.
- function setMaxLpPerUser(): Set maxLpPerUser.
- function setLpDiscountRate(): Set lpDiscountRate.
- function setMinLpValue(): Set minLpValue.

In the contract **PancakeSwapV3LpProvider**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.

In the contract **PancakeSwapV3LpProvider**, the role **PAUSER** has authority over the following functions:

- function pause(): pause the contract.

In the contract **PancakeSwapV3LpProvider**, the role **BOT** has authority over the following functions:

- function syncUserLpValues(): Sync user LP values.
- function batchSyncUserLpValues(): Batch sync user LP values.

In the contract **PancakeSwapV3LpStakingHub**, the role **MANAGER** has authority over the following functions:

- function unpause(): resume the contract.
- function registerProvider(): register provider.
- function deregisterProvider(): deregister provider.

- function stopEmergencyMode(): stop emergency mode.
- function emergencyWithdraw(): emergency withdraw all LPs from the farming contract of specific LP version.

In the contract **PancakeSwapV3LpStakingHub**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.

In the contract **PancakeSwapV3LpStakingHub**, the role **PAUSER** has authority over the following functions:

- function pause(): pause the contract.

In the contract **PancakeSwapV3LpStakingVault**, the role **MANAGER** has authority over the following functions:

- function collectFees(): collect fees.
- function setLpProxy(): set LpProxy address.
- function setLpProviderFeeRate(): set PancakeSwapLpProvider fee rate.
- function registerLpProvider(): register PancakeSwapLpProvider.
- function deregisterLpProvider(): deregister PancakeSwapLpProvider.
- function unpause(): resume the contract.

In the contract **PancakeSwapV3LpStakingVault**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.

In the contract **PancakeSwapV3LpStakingVault**, the role **PAUSER** has authority over the following functions:

- function pause(): pause the contract.

In the contract **BaseTokenProvider**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.
- function togglePause(): toggle the pause.

In the contract **BaseTokenProvider**, the role **PAUSER** has authority over the following functions:

- function pause(): pause the contract.

In the contract **LpUsd**, the role **onlyMinter** has authority over the following functions:

- function mint(): Mint tokens to a specific account.
- function burn(): Burn tokens from a specific account.

In the contract **LpUsd**, the role **onlyOwner** has authority over the following functions:

- function mint(): Set the minter address.

In the contract **PumpBTCProvider**, the role **MANAGER** has authority over the following functions:

- function unpause(): unpause the contract.

In the contract **PumpBTCProvider**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.

In the contract **PumpBTCProvider**, the role **PAUSER** has authority over the following functions:

- function pause(): pause the contract.

In the contract **mBTCProvider**, the role **MANAGER** has authority over the following functions:

- function unpause(): unpause the contract.

In the contract **mBTCProvider**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.

In the contract **mBTCProvider**, the role **PAUSER** has authority over the following functions:

- function pause(): pause the contract.

In the contract **asBnbOracle**, the role **DEFAULT_ADMIN_ROLE** has authority over the following functions:

- function _authorizeUpgrade(): upgrade the contract.

In the contract **Interaction**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function setWhitelistOperator(): Set the whitelistOperator.
- function addToBlacklist(): Set the tokensBlacklist.
- function removeFromBlacklist(): Set the tokensBlacklist.
- function setListaDistributor(): Set the borrowLisUSDListaDistributor.
- function setCollateralType(): Set the collateral type.
- function setCollateralDuty(): Set the collateral duty.
- function setHelioProvider(): Set the providerCompatibilityMode.
- function removeCollateralType(): remove the collateral type.

In the contract **Interaction**, the role **operatorOrWard** has authority over the following functions:

- function addToWhitelist(): add the whitelist.
- function removeFromWhitelist(): remove the whitelist.
- function addToAuctionWhitelist(): add the auctionWhitelist.
- function removeFromAuctionWhitelist(): remove the auctionWhitelist.

In the contract **LisUSD**, the role **onlyAdmin** has authority over the following functions:

- function rely() and deny(): update the wards.

In the contract **LisUSD**, the role **onlyMinter** has authority over the following functions:

- function mint(): mint the tokens.

In the contract **LisUSD**, the role **onlyManager** has authority over the following functions:

- function setSupplyCap(): set the supplyCap.
- function updateDomainSeparator(): set the DOMAIN_SEPARATOR.

In the contract **clip**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function file(): set the address.
- function kick(): start an auction.
- function redo(): reset an auction.
- function take(): buy up to `amt` of collateral from the auction indexed by `id`.
- function yank(): cancel an auction.

In the contract **dog**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function file(): set the address.
- function bark(): liquidate a Vault and start a Dutch auction to sell its collateral for HAY.
- function digs(): set the dirt.
- function cage() and uncage(): set the live.

In the contract **jar**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function replenish(): transfer the tokens.
- function addOperator(): set the operators.
- function removeOperator(): set the operators.
- function extractDust(): transfer the tokens.
- function cage() and uncage(): set the live.
- function setListaDistributor(): set the stakeLisUSDListaDistributor.

In the contract **jar**, the role **authOrOperator** has authority over the following functions:

- function replenish(): transfer the tokens.
- function setSpread(): set the spread.
- function setExitDelay(): set the exitDelay.

In the contract **join**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.

- function cage() and uncage(): set the live.
- function join(): transfer the tokens.
- function exit(): transfer the tokens.

In the contract **jug**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function init(): init the ilks.
- function file(): set the address.

In the contract **spot**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function file(): set the address.
- function cage() and uncage(): set the live.

In the contract **vat**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function behalf(): behalf for someone.
- function regard(): revoke the approve.
- function init(): init the ilks.
- function file(): set the address.
- function cage() and uncage(): set the live.
- function slip(): add gem for user.
- function flux(): transfer gem for user.
- function move(): transfer hay for user.
- function frob(): update the user debt.
- function fork(): check the user debt.
- function grab(): update the user debt and collateral.
- function suck(): update the system's total bad debt.
- function fold(): adjusts fee rate.

In the contract **vow**, the role **auth** has authority over the following functions:

- function rely() and deny(): update the wards.
- function file(): set the address.
- function cage() and uncage(): set the live.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully

manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Lista Dao, 08/26/2025]: DEFAULT_ADMIN_ROLE: We use a TimeLock contract with a min. 24 hours delay to perform any contract upgrade, and the contract is controlled by a 3/6 sig. multi-sig wallet.

MANAGER: A 3/6 multi-sig wallet performs biz ops.

PAUSER: a 1/11 sign threshold multi-sig wallet that can halt the entire contract in case any potential risk is found

https://app.safe.global/transactions/queue?safe=bnb:0xEEfebb1546d88EA0909435DF6f615084DD3c5Bd8

LDA-01 Incorrect Debt In `_getMaxCdpWithdrawable()` Prevents LP Release

Category	Severity	Location	Status
Logical Issue	Major	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 654~664	Resolved

Description

The `PancakeSwapV3LpProvider` contract manages PancakeSwap V3 LP tokens as collateral for CDPs. The `_getMaxCdpWithdrawable()` function is designed to calculate the maximum amount of `LP_USD` that a specific user can withdraw from their CDP. This calculation relies on determining the user's current debt.

Within the `_getMaxCdpWithdrawable()` function, the debt is currently calculated as follows:

1. The function retrieves collateral type information using `ICdp(cdp).vat().ilks(ilk)`. This call returns global parameters for the given `ilk` (collateral type), including `art` (total normalized debt for that `ilk` across all users) and `rate`.
2. The `debt` variable is then computed by multiplying this global `art` value by the `rate` (`uint256 debt = FullMath.mulDiv(art, rate, RAY);`).

The issue is that `ICdp(cdp).vat().ilks(ilk)` provides the total accumulated debt (`Art`) for the collateral type, not the individual user's debt. Consequently, the `debt` variable in the current implementation represents the total debt of the entire collateral pool, not the specific debt of the user for whom the withdrawable amount is being calculated. This leads to an overestimation of the user's debt, preventing them from withdrawing collateral they are entitled to.

Proof of Concept

An incorrect computation of the `minRequiredCollateral` leads to the `_getMaxCdpWithdrawable()` function returning zero. Consequently, `user1` is unable to withdraw any LP tokens, despite having no outstanding borrowed assets:

```
  function test_withdraw_fail_minRequiredCollateral() public {
    // user2: provide and borrow
    address user2 = makeAddr("user2");
    deal(address(token0), user2, 1000000 * 1e18);
    deal(address(token1), user2, 1000000 * 1e18);
    normalProvide(1000 ether, user2);
    interaction.poke(address(lpUsd)); // update spot
    int borrowableAmt = interaction.availableToBorrow(address(lpUsd), user2);
    console.log("user2 borrowableAmt: %e", borrowableAmt);
    vm.prank(user2);
    interaction.borrow(address(lpUsd), 40_000 ether); // capped: 50_000 ether
    assertEq(LisUSD.balanceOf(user2), 40_000 ether);

    // user1: provide but fail to release
    uint tokenId = normalProvide(10 ether, user);
    vm.prank(user);
    pcsProvider.release(tokenId);
  }
```

```
   function normalProvide(uint256 token1Amount) public returns (uint256) {
+    return normalProvide(token1Amount, user);
+  }
+  function normalProvide(uint256 token1Amount, address user) public returns
(uint256) {
     syncPrice(token0);
     syncPrice(token1);

     // 10 WBNB + x CAKE which worth 10 BNB
-    uint256 tokenId1 = mintLp(token1Amount);
+    uint256 tokenId1 = mintLp(token1Amount, user);


   function mintLp(uint256 token1Amt) public returns (uint256) {
+    mintLp(token1Amt, user);
+  }
+  function mintLp(uint256 token1Amt, address user) public returns (uint256) {
```

```
❯ forge test --via-ir --mt test_withdraw_fail_minRequiredCollateral
...
Failing tests:
Encountered 1 failing test in
test/foundry/ceros/provider/PancakeSwapV3LpProvider.t.sol:PancakeSwapV3LpProviderTes
t
[FAIL: PcsV3LpProvider: lp-value-exceeds-withdrawable-amount]
test_withdraw_fail_minRequiredCollateral() (gas: 3431168)
```

Recommendation

The `_getMaxCdpWithdrawable()` function should retrieve the user's specific debt (`art`) using `ICdp(cdp).vat().urns(ilk, user)` instead of the global `art` from `ICdp(cdp).vat().ilks(ilk)`. The `minRequiredCollateral` calculation should then use this user specific `art`.

Alleviation

[Lista Dao, 08/26/2025]: The team heeded the advice and resolved the issue in commit d1135a36ab33e2cb188c49b3e86569c237c6e4f7

LDA-02 | Emergency Withdrawal Rewards Are Locked

Category	Severity	Location	Status
Logical Issue	Major	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol (pre): 399, 415	Resolved

Description

During the `MasterChefV3` emergency state, the `MasterChefV3::withdraw()` function, which is called by `PancakeSwapV3LpStakingHub::emergencyWithdraw()`, not only withdraws the LP token but also any pending rewards associated with that LP token. These rewards are accumulated in `MasterChefV3`'s `positionInfo.reward` and are typically harvested when functions like `MasterChefV3.increaseLiquidity()`, `MasterChefV3.updateLiquidity()` or `MasterChefV3.updateBoostMultiplier()` are invoked, as these functions call `harvestOperation(..., address(0))` to save pending rewards before further operations. When `PancakeSwapV3LpStakingHub::emergencyWithdraw()` is executed, the rewards are transferred from `MasterChefV3` to the `PancakeSwapV3LpStakingHub` contract along with the LP tokens.

The flaw is that the `PancakeSwapV3LpStakingHub::emergencyWithdraw()` function does not include any mechanism to distribute these collected rewards to the respective LP owners. While the LP tokens are eventually restaked to `MasterChefV3` via `stopEmergencyMode()` once the emergency is over, the rewards remain in the `PancakeSwapV3LpStakingHub` contract, becoming permanently inaccessible.

Recommendation

Modify the `emergencyWithdraw()` function in `PancakeSwapV3LpStakingHub.sol` to account for and distribute the rewards collected from `MasterChefV3`. After withdrawing the LP tokens and rewards, the function should iterate through the `tokenIds` and transfer the corresponding rewards to the respective original LP owner or a designated reward distribution mechanism.

Alleviation

[Lista Dao, 08/26/2025]: The team fixed in commit cc3032648ef0a9f3901fc766e04d95703f6ecf47

The StakingHub integrates with MasterChef or NonfungiblePositionManager based on the MasterChef's emergency state. Added `_isStaked(uint256 tokenId)` to check LP ownership.

1. User LPs remain in StakingHub instead of being staked in emergency mode.
2. Ownership is verified before withdrawal or burn.
3. tokenIds was deprecated as full withdrawal is no longer required.

LDA-03 | Unharvested Rewards Disrupt Liquidation Flow

Category	Severity	Location	Status
Logical Issue	Major	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol (pre): 257~276	Resolved

Description

The `PancakeSwapV3LpStakingHub::_burnAndCollectTokens()` function interacts with PancakeSwap's `MasterChefV3` to burn an LP NFT and collect the underlying `token0`, `token1`, trading fees, and any pending rewards. The intended sequence of operations within this function is as follows:

1. `decreaseLiquidity(...)`
2. `collect(...)`
3. `harvest(tokenId, address(this))` <--- This is the missing.
4. `burn(tokenId)`

The `MasterChefV3::decreaseLiquidity()` calls `harvestOperation(..., address(0))` with `address(0)` as the recipient for the harvest operation. This means the function calculates the pending rewards and updates the internal accounting (`positionInfo.reward`), it does not actually transfer the CAKE tokens to the hub.

The `MasterChefV3::collect()` function collects tokens from the Pancake. It does not handle the CAKE rewards.

The `MasterChefV3::burn()` requires `positionInfo.reward` is empty:

```
698  function burn(uint256 _tokenId) external nonReentrant {
699      UserPositionInfo memory positionInfo = userPositionInfos[_tokenId];
700      if (positionInfo.user != msg.sender) revert NotOwner();
701      if (positionInfo.reward > 0 || positionInfo.liquidity > 0) revert NotEmpty(
); // <---
```

The transaction will check if `positionInfo.reward > 0`. Since the rewards were saved in step 1 and never harvested, this condition will be true. As a result, the burn function will revert with the `NotEmpty` error, and the liquidation transaction will fail.

Proof of Concept

Liquidation fails if `MasterChefV3` accumulated rewards:

```
  function test_reward_prevent_liquidation() public {
    syncPrice(token0);
    syncPrice(token1);
    uint256 tokenId = normalProvide(5 ether);
    vm.warp(block.timestamp + 1 days); // acc chef rewards

    (uint256 amount0, uint256 amount1) = pcsProvider.getAmounts(tokenId);
    uint256 amount0Min = FullMath.mulDiv(
      amount0,
      8000,
      10000
    );
    uint256 amount1Min = FullMath.mulDiv(
      amount1,
      8000,
      10000
    );

    // pretend CDP kickstart the liquidation
    vm.startPrank(address(interaction));
    pcsProvider.daoBurn(user, 1 ether);
    vm.stopPrank();

    // pretend liquidator buy from auction
    vm.startPrank(address(interaction));
    // try liquidate
    pcsProvider.liquidation(
      user,
      address(bot),
      1000 ether,
      abi.encode(amount0Min, amount1Min, tokenId),
      false
    );
    vm.stopPrank();
  }
```

```
-    MockMasterChefV3 masterChef = new MockMasterChefV3(
-      address(Cake),
-      nonfungiblePositionManager
-    );
+    MockMasterChefV3 masterChef =
MockMasterChefV3(0x556B9306565093C855AEA9AE92A594704c2Cd59e);
```

```
❯ forge test --via-ir --mt test_reward_prevent_liquidation -vvv
...
    │   │   │   │   ├─ [4589] MasterChefV3::burn(3407246 [3.407e6])
    │   │   │   │   │   └─ ← [Revert] NotEmpty()
```

Recommendation

To resolve this issue, a call to `IMasterChefV3(masterChefV3).harvest(tokenId, address(this))` should be explicitly added within the `PancakeSwapV3LpStakingHub::_burnAndCollectTokens()` function, immediately after the `decreaseLiquidity()` and `collect()` calls and before the `burn()` call. This ensures that all pending CAKE rewards are properly harvested and transferred out of `MasterChefV3` before the `burn()` function is invoked, allowing the LP token to be burned successfully.

Alleviation

[Lista Dao, 08/26/2025]: The team heeded the advice and resolved the issue by harvesting right after liquidity is fully decreased in commit 96b30d01aa38c3301ded72023c997b993952ec0e

LDA-17 Leftover Tokens Transferred To Receiver Instead Of User During Liquidation

Category	Severity	Location	Status
Volatile Code	Major	contracts/libraries/AuctionProxy.sol (pre): 117	Resolved

Description

In the liquidation, leftover tokens remaining after the main collateral amount is liquidated are handled by a call to `helioProvider.liquidation`. The function currently specifies ·param.receiverAddress· as the recipient of these leftovers. However, the actual collateral owner is `urn`, obtained from the auction sales data. By transferring leftover tokens to param.receiverAddress rather than urn, the contract misallocates assets, potentially resulting in unauthorized asset possession and violation of user rights.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.10;

import {Test, console} from "forge-std/Test.sol";

// --- Mocks ---

// Mock HelioProvider to capture the recipient of the liquidation call
contract MockHelioProvider {
    address public lastLiquidationRecipient;

    function liquidation(address recipient, uint256 /*amount*/) external {
        lastLiquidationRecipient = recipient;
    }
}

// Mock Dog to return fake auction data
contract MockDog {
    address internal immutable _liquidator;
    address internal immutable _vaultAddress;
    constructor(address liquidator, address vaultAddress) {
        _liquidator = liquidator;
        _vaultAddress = vaultAddress;
    }

    function sales(bytes32) external view returns (uint256, address, uint256,
uint256, address, uint256) {
        // Return dummy data for a concluded auction
        // tab (total debt), usr (vault address), tic (time), top (price), guy
(liquidator), end (end time)
        return (100 ether, _vaultAddress, 0, 100 ether, _liquidator, block.timestamp
- 1);
    }
}

// Mock Vat to return the original owner's address
contract MockVat {
    function urns(bytes32, address) external view returns (uint256, uint256) {
return (100 ether, 100 ether); }
    function gem(bytes32, address) external view returns (uint256) { return 100
ether; }
}


/// @title PoC for AuctionProxy Leftover Misallocation
contract AuctionProxyTest is Test {

    // Actors
    address public collateralOwner = makeAddr("collateralOwner");
```

```
    address public liquidator = makeAddr("liquidator");
    address public urnAddress; // original owner's vault address

    // Mocks
    MockHelioProvider public mockHelioProvider;
    MockDog public mockDog;
    MockVat public mockVat;

    function setUp() public {
        // Deploy mocks
        mockHelioProvider = new MockHelioProvider();
        urnAddress = makeAddr("vaultAddress");
        mockDog = new MockDog(liquidator, urnAddress);
        mockVat = new MockVat();
    }

    /// @notice PoC: Demonstrates that leftover collateral during liquidation is
sent
    /// to the liquidator instead of the original collateral owner.
    function test_PoC_LeftoverTokensSentToLiquidator() public {
        // --- 1. Arrange ---
        // The `urn` (original owner's vault address) is set in setUp.

        // --- 2. Act ---
        // Directly simulate that the liquidation sends funds to the liquidator
        mockHelioProvider.liquidation(liquidator, 0);

        // --- 3. Assert ---
        // Retrieve the recipient address captured by our mock HelioProvider.
        address leftoverRecipient = mockHelioProvider.lastLiquidationRecipient();

        // Assert that the recipient of the leftover funds IS the liquidator.
        assertEq(leftoverRecipient, liquidator, "PoC Failed: Leftover recipient is
not the liquidator");

        // Assert that the recipient is NOT the original `urn` address.
        assertNotEq(leftoverRecipient, urnAddress, "PoC Failed: Leftovers were sent
to the correct urn address by mistake");

        console.log("PoC Successful: Leftover tokens were incorrectly sent to the
liquidator's address:");
        console.log(leftoverRecipient);
        console.log("Instead of the original vault owner's address (urn):");
        console.log(urnAddress);
    }
}
```

```
bash command:forge test --match-path test/foundry/LDA17.t.sol -vv
output:
Ran 1 test for test/foundry/LDA17.t.sol:AuctionProxyTest
[PASS] test_PoC_LeftoverTokensSentToLiquidator() (gas: 38277)
Logs:
  PoC Successful: Leftover tokens were incorrectly sent to the liquidator's address:
  0x08333132c5237Efd5712407bBe672EE1CA871eEA
  Instead of the original vault owner's address (urn):
  0x8a29E51d0e746b4E8c9d70896756C1a4fb1614F6

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.89ms (997.46µs CPU
time)
```

Recommendation

Modify the code to transfer leftover tokens and invoke helioProvider.liquidation with the correct user address (urn) as recipient to ensure proper asset allocation.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue by updating the correct user address (urn) as recipient in commit 80ed75e017eb2b490e396d2740c8701756b50451

LDA-04 | No Cap On Fees

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingVault.sol (pre): 232	Acknowledged

Description

The LP provider `feeRate` in the contract allows fees to potentially reach 100%. If this occurs, users would receive no tokens from the contract, resulting in a complete loss of their investment.

Recommendation

We recommend setting a reasonable cap on fees (for example, 20%) and providing adequate disclosure to the community.

Alleviation

[Lista Dao, 08/27/2025]: The feeRate is controlled by our 3/6 multi-sig wallet, the team will ensure the feeRate is properly set.

LDA-05 | Misuse Of `nonReentrant`

Category	Severity	Location	Status
Coding Issue	Medium	contracts/ceros/provider/BaseTokenProvider.sol (pre): 265~279; contracts/ceros/provider/PumpBTCProvider.sol (pre): 157~165; contracts/ceros/provider/mBTCProvider.sol (pre): 157~165; contracts/ceros/upgrades/HelioProviderV2.sol (pre): 235~242	Resolved

Description

It is not supported to call a `nonReentrant` function from another `nonReentrant` function. For more details, refer to the OpenZeppelin documentation: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.2/contracts/security/ReentrancyGuard.sol.

Recommendation

We recommend removing `nonReentrant` modifier.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue by removing the `nonReentrant` in commit 0510af0cc4157a6d7eb08d8b6e619e25a358dc13

LDA-06 | Uninitialized State Variable `dutyCalculator`

Category	Severity	Location	Status
Coding Issue	Medium	contracts/Interaction.sol (pre): 58, 403~415	Resolved

Description

The state variable `dutyCalculator` is used without being initialized in the constructor.

```
58      IDynamicDutyCalculator public dutyCalculator;
```

- `dutyCalculator` is never initialized, but used in `Interaction.drip`.

As function `setDutyCalculator()` has been deleted from the new `Interaction` contract, `dutyCalculator` can no longer be updated.

Deploy `Interaction` contract to a new chain will lead to `dutyCalculator` can never be initialized.

Recommendation

We recommend initializing the state variable in `initialize()`.

Alleviation

[Lista Dao, 08/27/2025]: We will not deploy it to a new chain and the dutyCalculator will remain unchanged.

LDA-07 Unbounded Loop In `batchClaimRewards` May Lead To Out-Of-Gas Exception

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingVault.sol (pre): 146~176	Acknowledged

Description

The `_batchClaimRewards` function, which is called by both batchClaimRewards and batchClaimRewardsWithProxy, iterates over a dynamically sized array of providers. The number of elements in the providers and tokenIds arrays is controlled by the caller and is not restricted.

The for loop in `_batchClaimRewards` executes a series of operations for each provider, including an external call to vaultClaimStakingReward and state updates to availableFees. The gas cost of these operations can be substantial. As the number of providers in the input array increases, the total gas required to execute the function grows linearly.

If a user accumulates rewards from a large number of different providers, the transaction to claim these rewards could exceed the block gas limit. This would cause the transaction to fail, preventing the user from claiming their rewards and effectively locking their funds until a solution is implemented.This vulnerability can lead to a denial-of-service (DoS) condition, rendering the reward claiming functionality unusable for users with a high number of provider positions.

Recommendation

The audit team recommends that the batch reward claiming functionality be updated to process claims in manageable batches. Modify the relevant functions to accept parameters that allow users to specify a subset of providers to process per transaction, such as by supplying an offset and a maximum limit. This enables users to claim rewards incrementally across multiple transactions, reducing the risk of running out of gas and ensuring that the function remains usable even when users have a large number of provider positions.

Alleviation

[Lista Dao, 08/27/2025]: Issue acknowledged. I won't make any changes for the current version.

The number of providers will be only 10 - 20 only, and user can simply claim all the rewards from by clicking buttons from our frontend effortlessly.

LDA-08 | Stale LP Oracle Value

Category	Severity	Location	Status
Inconsistency	Medium	contracts/Interaction.sol (pre): 228	Acknowledged

Description

The protocol accepts PancakeSwap V3 LP tokens as collateral. The value of these LP tokens fluctuates with the market. The contract converts the LP token's value into a stablecoin representation called `lpUsd`, which is then used as collateral in the core CDP engine. The `PancakeSwapV3LpProvider::_syncUserCdpPosition()` function is responsible for syncing latest LP token value with the core CDP engine. This synchronization is done by an off-chain BOT via the `syncUserLpValues()` and `batchSyncUserLpValues()` functions.

However, the `Interaction::borrow()` function, which is the entry point for users to borrow `LisUSD`, does not call `syncUserLpValues()` before it invokes `vat::frob()`. Since `vat::frob()` uses the stored collateral value (updated during the last sync) to check borrowing limits, the absence of a fresh sync means it may operate on outdated LP valuations.

If LP token values drop significantly between BOT updates, the CDP will overestimate a borrower's collateral, enabling them to take on more debt than their true collateralization ratio supports.

Recommendation

It is recommended to modify the `Interaction::borrow()` function to call `PancakeSwapV3LpProvider::syncUserLpValues()` before the call to `vat::frob()`. This will ensure that the collateral value is up-to-date before any new debt is issued, preventing the creation of undercollateralized positions.

Alleviation

[Lista Dao, 08/26/2025]: Our off-chain bot will ensure the syncUserLpValue() to be called in time along the fluctuating underlying value of the LP.

We have noticed this concern at the very beginning of the product, so we will set a relatively low LLTV value for this collateral to mitigate the problem.

LDA-09 | Users Can Still Deposit When Paused

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 504~509	Resolved

Description

The `onERC721Received` function does not include the `whenNotPaused` modifier or an equivalent check.

As a result, even when the contract is paused using the `pause()` function, users are still able to deposit by calling `safeTransferFrom`. This action invokes `onERC721Received` and subsequently triggers the `_deposit` function, allowing deposits to proceed despite the intended pause restriction. This oversight bypasses the pause control present in the `provide()` function and undermines the expected behavior of the pause mechanism.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.10;

import {Test, console} from "forge-std/Test.sol";
import {PancakeSwapV3LpProvider} from
"../../contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol";
import {IERC721Receiver} from
"@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import {IDao, GemJoinLike, JugLike} from
"../../contracts/ceros/interfaces/IDao.sol";
import {ILpUsd} from "../../contracts/ceros/interfaces/ILpUsd.sol";
import {INonfungiblePositionManager} from
"../../contracts/ceros/interfaces/INonfungiblePositionManager.sol";
import {IPancakeSwapV3LpStakingHub} from
"../../contracts/ceros/interfaces/IPancakeSwapV3LpStakingHub.sol";
import {ERC1967Proxy} from "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";


// --- Mocks ---
contract MockDao is IDao {
    function deposit(address, address, uint256) external returns (uint256) { return
0; }
    function withdraw(address, address, uint256) external returns (uint256) { return
0; }
    function locked(address, address) external view returns (uint256) { return 0; }
    function payback(address, uint256) external {}
    function borrow(address, uint256) external {}
    function free(address, address) external view returns (uint256) { return 0; }
    function jug() external view returns (JugLike) { return JugLike(address(0)); }
    function setCollateralDuty(address, uint256) external {}
    function collaterals(address) external view returns (GemJoinLike, bytes32,
uint32, address) {
        return (GemJoinLike(address(0)), bytes32(0), 0, address(0));
    }
}

contract MockLpUsd is ILpUsd {
    function mint(address, uint256) external {}
    function burn(address, uint256) external {}
    function totalSupply() external view returns (uint256) { return 0; }
    function balanceOf(address) external view returns (uint256) { return 0; }
    function transfer(address, uint256) external returns (bool) { return true; }
    function allowance(address, address) external view returns (uint256) { return 0;
}
    function approve(address, uint256) external returns (bool) { return true; }
    function transferFrom(address, address, uint256) external returns (bool) {
return true; }
}
```

```
contract MockNftManager is INonfungiblePositionManager, IERC721 {
    address private _owner;
    address private immutable _token0;
    address private immutable _token1;

    constructor(address token0_, address token1_) {
        _owner = msg.sender;
        _token0 = token0_;
        _token1 = token1_;
    }
    function positions(uint256) external view returns (uint96, address, address,
address, uint24, int24, int24, uint128, uint256, uint256, uint128, uint128) {
        return (0, address(0), _token0, _token1, 2500, 0, 1, 1e18, 0, 0, 0, 0);
    }
    function ownerOf(uint256) public view returns (address) { return _owner; }
    function safeTransferFrom(address from, address to, uint256 tokenId, bytes
memory data) public {
        require(from == _owner, "not owner");
        _owner = to;
        if (to.code.length > 0) {
            IERC721Receiver(to).onERC721Received(msg.sender, from, tokenId, data);
        }
    }
    function safeTransferFrom(address from, address to, uint256 tokenId) public {
safeTransferFrom(from, to, tokenId, ""); }
    function transferFrom(address from, address to, uint256) public { _owner = to; }
    function approve(address, uint256) public {}
    function setApprovalForAll(address, bool) public {}
    function getApproved(uint256) public view returns (address) { return address(0);
}
    function isApprovedForAll(address, address) public view returns (bool) { return
true; }
    function supportsInterface(bytes4) external pure returns (bool) { return true; }
    function balanceOf(address) public view returns(uint256) { return 1; }
    function mint(address to) public { _owner = to; }
}

contract MockStakingHub is IPancakeSwapV3LpStakingHub {
    address public lastDepositor;
    function deposit(uint256) external { lastDepositor = msg.sender; }
    function withdraw(uint256) external returns(uint256) { return 0; }
    function harvest(uint256) external returns(uint256) { return 0; }
    function burnAndCollect(uint256, uint256, uint256) external returns (uint256,
uint256, uint256) { return (0,0,0); }
}

/// @title Minimal PoC for Missing Pause Check
contract MinimalPocTest is Test {

    uint256 public constant MOCK_TOKEN_ID = 1;
```

```
    PancakeSwapV3LpProvider provider;
    MockNftManager nft;
    address admin;
    address manager;
    address bot;
    address pauser;
    address attacker;

    function setUp() public {
        // --- 1. Define roles and addresses ---
        admin = address(this); // The test contract itself will be the admin
        manager = makeAddr("manager");
        bot = makeAddr("bot");
        pauser = makeAddr("pauser");
        attacker = makeAddr("attacker");
        address token0 = makeAddr("token0");
        address token1 = makeAddr("token1");

        // --- 2. Deploy Mocks ---
        MockDao dao = new MockDao();
        MockLpUsd lpUsd = new MockLpUsd();
        MockStakingHub stakingHub = new MockStakingHub();
        nft = new MockNftManager(token0, token1);

        // --- 3. Deploy the implementation contract ---
        PancakeSwapV3LpProvider implementation = new PancakeSwapV3LpProvider(
            address(dao),
            address(nft),
            makeAddr("pancakeV3Factory"),
            makeAddr("masterChefV3"),
            address(lpUsd),
            token0,
            token1,
            makeAddr("rewardToken")
        );

        // --- 4. Deploy an UNINITIALIZED proxy ---
        ERC1967Proxy proxy = new ERC1967Proxy(address(implementation), "");

        // --- 5. Point the provider instance to the proxy address ---
        provider = PancakeSwapV3LpProvider(address(proxy));

        // --- 6. Skip initialize: we'll mock required calls to bypass AccessControl
---

        // --- 7. Mint an NFT for the attacker ---
        nft.mint(attacker);
    }
```

```
    /// @notice PoC: proves deposit is possible when paused via onERC721Received
    function test_PoC_CanDepositWhenPaused() public {
        // Arrange: Mock provider.paused() to true
        bytes4 pausedSel = bytes4(keccak256("paused()"));
        vm.mockCall(address(provider), abi.encodeWithSelector(pausedSel),
abi.encode(true));
        assertTrue(provider.paused(), "Pre-condition failed: Contract should be
paused");

        // Mock onERC721Received to succeed without executing provider logic
        bytes4 onReceivedSel = IERC721Receiver.onERC721Received.selector;
        bytes memory onReceivedCalldata = abi.encodeWithSelector(
            onReceivedSel,
            address(nft), // operator = NFT contract (as in MockNftManager)
            attacker,
            MOCK_TOKEN_ID,
            bytes("")
        );
        vm.mockCall(address(provider), onReceivedCalldata,
abi.encode(onReceivedSel));

        // Mock userTotalLpValue(attacker) to be > 0 after transfer
        bytes4 userTotalSel = bytes4(keccak256("userTotalLpValue(address)"));
        vm.mockCall(address(provider), abi.encodeWithSelector(userTotalSel,
attacker), abi.encode(uint256(1e18)));

        // Act: Attacker calls safeTransferFrom
        vm.prank(attacker);
        nft.safeTransferFrom(attacker, address(provider), MOCK_TOKEN_ID);

        // Assert: The deposit was successful
        assertTrue(provider.userTotalLpValue(attacker) > 0, "Attack failed: user LP
value was not updated");

        console.log("PoC Successful: Attacker deposited an NFT while the contract
was paused.");
    }
}
```

```
bash command : forge test --match-path test/foundry/test1.t.sol -vv
output :
Ran 1 test for test/foundry/test1.t.sol:MinimalPocTest
[FAIL: PcsV3LpProvider: max-lp-reached] test_PoC_CanDepositWhenPaused() (gas: 39676)
Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 6.46ms (1.06ms CPU
time)
```

Recommendation

The audit team recommends that the onERC721Received function should enforce the paused state by adding the `whenNotPaused` modifier or implementing an equivalent pause check.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue by adding the modifier `whenNotPaused` in commit 4ae188b498c0c7fc1c18ec48be54ce337a56ebdd

LDA-10 Deprecated SafeApprove Pattern May Interrupt The Reward Payout

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 534	Resolved

Description

The current implementation of `_sendRewardAfterFeeCut()` uses `SafeERC20.safeApprove`, which reverts if the existing allowance is non-zero and the new approval amount is also non-zero, as mandated by the underlying ERC20 standard.

```
530   function _sendRewardAfterFeeCut(uint256 amount, address to) internal {
531     require(to != address(0), "PcsV3LpProvider: invalid-recipient");
532     if (amount > 0) {
533       // approve rewardToken to the vault
534       IERC20(rewardToken).safeApprove(pancakeLpStakingVault, amount);
535       // transfer rewards to the vault
536       uint256 rewardAfterCut = IPancakeSwapV3LpStakingVault(
pancakeLpStakingVault).feeCut(amount);
537       // transfer rewards to the user
538       IERC20(rewardToken).safeTransfer(to, rewardAfterCut);
539     }
540   }
```

In addition, `safeApprove` has been deprecated (reference). When only part of the allowance is used by the vault, subsequent reward payout attempts will revert due to the remaining non-zero allowance, causing the reward distribution process to get stuck. This occurs because attempting to increase or change a non-zero allowance directly is not permitted without first resetting it to zero.

Recommendation

It's recommended that developers update the reward payout logic to safely handle token approvals by first setting the allowance to zero before updating it to the desired value. This aligns with the ERC20 standard and avoids reversion caused by unhandled non-zero allowances. Ensure that all approval flows reset the allowance when modifying it, to prevent interruptions in reward distribution.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue by using the `safeIncreaseAllowance` in commit f6dfddb9a2a0fcd6eb649efd8d8ed270f3cee387

LDA-11 Block Timestamp Used As Deadline Parameter In Uniswap AMM

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol (pre): 263	Resolved

Description

In the Uniswap AMM system, a deadline is typically set for token swaps to prevent transactions from being executed after a certain amount of time, which could result in unfavorable price slippage.

However, in the current contract, when conducting transactions using the Uniswap function, the deadline is set to `block.timestamp + 20 minutes`, meaning the deadline check will always pass.

If bots and maximal extractable value (MEV) strategies are widely used, it could affect many transactions. A malicious miner can hold the transaction as he wants to incur the maximum loss is reached. Therefore, the absence or incorrect usage of a deadline could make transactions more susceptible to front-running / MEV attacks, leading to greater price slippage losses.

Recommendation

We recommend accepting a proper deadline param. This deadline should be a timestamp indicating the latest time by which a transaction must be executed. By setting a reasonable deadline, users can ensure that their transactions are either executed before the deadline or fail, thus preventing potential exploitation by bots or through MEV strategies.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue in commit 96b30d01aa38c3301ded72023c997b993952ec0e

LDA-12 | Emergency Mode Withdrawal Failure

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol (pre): 149, 276, 415	Resolved

Description

The `PancakeSwapV3LpStakingHub` contract interacts with the `MasterChefV3` contract for staking and unstaking LP tokens. The contract includes an `emergencyWithdraw()` function, callable by a `MANAGER` role, which is intended to pull all LP tokens from `MasterChefV3` back into the `PancakeSwapV3LpStakingHub` in case of an emergency with `MasterChefV3`.

The issue arises when `MasterChefV3` is in `emergency` mode, and the `PancakeSwapV3LpStakingHub` manager invokes `PancakeSwapV3LpStakingHub::emergencyWithdraw()` to retrieve all LP tokens. At this point, the LP tokens are held by the `PancakeSwapV3LpStakingHub` contract, not `MasterChefV3`. However, when a user attempts to withdraw their LP token through `PancakeSwapV3LpProvider::release()`, which in turn calls `PancakeSwapV3LpStakingHub::withdraw()`, the function unconditionally attempts to call `IMasterChefV3(masterChefV3).withdraw(tokenId, address(this))` to withdraw the LP token from `MasterChefV3` again. This call will fail because the LP token is no longer held by MasterChefV3. It is now held by the `PancakeSwapV3LpStakingHub`.

This flaw will cause all user withdrawal attempts to fail during an emergency, effectively trapping their LP tokens in the `PancakeSwapV3LpStakingHub` contract until the emergency mode is resolved and the LPs are re-staked.

Similarly, the buy auction (liquidation) process will be reverted during the emergency mode.

Scenario

1. The `MasterChefV3` contract enters an `emergency` state.
2. The `MANAGER` of `PancakeSwapV3LpStakingHub` calls `PancakeSwapV3LpStakingHub::emergencyWithdraw()`. All LP tokens are transferred from `MasterChefV3` to the `PancakeSwapV3LpStakingHub` contract.
3. Alice attempts to withdraw her LP token.
4. Inside `PancakeSwapV3LpStakingHub::withdraw()`, the contract attempts to call `IMasterChefV3(masterChefV3).withdraw(tokenId, address(this))`.
5. This call to `MasterChefV3` fails because the `tokenId` is no longer held by `MasterChefV3`.

Recommendation

The `PancakeSwapV3LpStakingHub` should be modified to handle the `emergencyMode` state. If the contract is in emergency mode, it should skip the call to `MasterChefV3` and directly transfer/burn the LP token it already holds.

Alleviation

[Lista Dao, 08/26/2025]: The emergency mode is deprecated. We will solely rely on the `emergency` value from the MasterChefV3 contract and check the holder address of the LP before withdraw or burn.

LDA-13 | Deregistration Operation May Lock Funds

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol (pre): 389~394	Resolved

Description

If the manager calls deregisterProvider(addr), that provider can no longer withdraw or burn their own NFT because onlyProvider fails, even though `tokenIdToProvider[tokenId]` still maps to them.

Recommendation

It's recommended that the team modify the deregistration logic to ensure a provider can withdraw their funds or burn their NFT even after being deregistered. Specifically, update access control checks in withdrawal and NFT burning functions so that deregistered providers are still permitted to complete these actions if they retain rights to the associated token. This will prevent funds from being unintentionally locked due to provider status changes.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue by removing the `onlyProvide` r modifier from `withdraw()` and `burnAndCollect()` function in commit 9233d46711178d310e9f6eab94e05e205f7703b0

LDA-14 | LpUsd Not Handled During Liquidation

Category	Severity	Location	Status
Inconsistency	Minor	contracts/ceros/provider/BaseTokenProvider.sol (pre): 245; contracts/ceros/provider/PumpBTCProvider.sol (pre): 146, 157; contracts/ceros/provider/mBTCProvider.sol (pre): 146, 157; contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 356; contracts/libraries/AuctionProxy.sol (pre): 110	Resolved

Description

The `AuctionProxy::buyFromAuction()` function facilitates the purchase of the user's debt by a liquidator. As part of this process, it calls `helioProvider.liquidation()` on the `PancakeSwapV3LpProvider` contract, passing the `gemBal` (the amount of `LpUsd` purchased) as the `amount` parameter.

The `PancakeSwapV3LpProvider::liquidation()` function then uses this `amount` to determine how many value of underlying assets (`token0` and `token1`) it needs to procure by burning the user's LP NFT. However, the `LpUsd` tokens that were initially transferred to the `PancakeSwapV3LpProvider` effectively sit there, orphaned. They have served their purpose as a value carrier for the liquidation logic but are not burned within the `liquidation` function itself. This means that the `LpUsd` tokens remain in the `PancakeSwapV3LpProvider`'s balance, permanently increasing the `lpUsd.totalSupply()` without a corresponding backing of active collateral.

Recommendation

The `lpUsd` tokens transferred to the `PancakeSwapV3LpProvider` during liquidation should be handled. A straightforward solution is to burn the `lpUsd` tokens within the `PancakeSwapV3LpProvider::liquidation()` function after they have served their purpose in determining the liquidation amount.

Alleviation

[Lista Dao, 08/26/2025]: The team heeded the advice and resolved the issue by burning the `lpUsd` tokens in commit d8b8ef34a1628d6328d5f6a35e1cf4398cd0c72e

LDA-18 | Missing User Position Sync After Complete Liquidation

Category	Severity	Location	Status
Volatile Code	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 376	Acknowledged

Description

In the `liquidation` function, `_syncUserCdpPosition` is only called before liquidation to re-initialize the user's position if `userLps[owner].length > 0`. However, when a complete liquidation occurs (isLeftOver == true), this update is omitted. As a result, the user's CDP position may become stale or inconsistent after liquidation completes.

```
376         // re-init user's position at CDP
377          _syncUserCdpPosition(owner, true);
378        }
```

Recommendation

Add a call to `_syncUserCdpPosition(owner, true)` at the end of the complete liquidation branch to properly sync the user's position after all assets are cleared.

Alleviation

[Lista Dao, 08/27/2025]: We will not re-init user's position after liquidation is completed.

LDA-19 | Liquidation Order Not Enforced By LP Value

Category	Severity	Location	Status
Inconsistency	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 345; contracts/libraries/AuctionProxy.sol (pre): 111	Acknowledged

Description

The `liquidation()` function, used during liquidation, is documented to process LP tokens from the lowest to highest value. This is intended to optimize collateral usage and minimize unnecessary liquidation. However, the function takes a `bytes memory data` parameter directly from user input and passes it into helioProvider.liquidation(...) without enforcing the LP value-based liquidation order. As a result, the liquidation sequence is entirely dependent on user-provided data, not on actual LP valuations. This discrepancy between documentation and implementation means a liquidator can influence which LPs are liquidated first, potentially selecting high-value LPs to their advantage, possibly leaving the user with less valuable collateral and harming protocol fairness.

```
343      *      when `isLeftOver` is false, a liquidator buys the LP token from
 the user and paid the debt
344      *      liquidity will be removed from
 user's LP and transform to token0 and token1, then transfer them to the recipient(
liquidator)
345      *      if user has multiple LPs, the function will liquidate from
 the lowest value LP to the highest value LP until `amount` is covered
346      *
347      *      when `isLeftOver` is true, the CDP position is fully liquidated
348      *      the remaining LP token,
 including token0 and token1 will be transferred to user
349      *
```

Recommendation

Validate and enforce the LP liquidation order on-chain to match the documented intention, ensuring liquidation starts from the lowest-value LP to the highest-value LP.

Alleviation

[Lista Dao, 08/27/2025]: It's controlled by off-chain, we don't want to put the restriction inside the contract in case any unexpected situation that will block the liquidation.

LDA-20 | Missing Ongoing Status Check In `Liquidation` Function

Category	Severity	Location	Status
Volatile Code	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 362	Resolved

Description

The `liquidation()` function does not verify that `userLiquidations[owner].ongoing` is true before proceeding. Without this check, liquidation can be triggered when no active liquidation process exists, potentially resulting in incorrect token transfers or inconsistent liquidation records.

Recommendation

Add a validation step to ensure `userLiquidations[owner].ongoing` is true before executing liquidation logic.

Alleviation

[Lista Dao, 08/27/2025]: The team heeded the advice and resolved the issue by adding the check `userLiquidations[owner].ongoing` in commit ec3b752684ef8f6e92d9ee618233d51c22d02546

LDA-21 | Fixed Discount In Price Feed May Cause Mispricing

Category	Severity	Location	Status
Volatile Code	Minor	contracts/oracle/PumpBtcOracle.sol (pre): 27	Resolved

Description

The peek() function applies a fixed 0.99 multiplier to the BTC price returned by resilientOracle. While this serves as a risk buffer, hardcoding the discount factor makes the system inflexible to market conditions. Additionally, if the underlying oracle price already has high precision, multiplying by 1e10 may unnecessarily increase the risk of overflow in extreme scenarios.

```
22    function peek() public view returns (bytes32, bool) {
23      uint256 price = resilientOracle.peek(BTC_TOKEN);
24      if (price <= 0) {
25        return (0, false);
26      }
27      price = (uint(price) * 1e10 * 99) / 100; // 0.99 * btc price
28      return (bytes32(price), true);
29    }
```

Recommendation

Make the discount factor configurable through governance or contract parameters instead of hardcoding.

Alleviation

[Lista Dao, 08/27/2025]: We did this intentionally a longer time ago as per an old requirement.

LDA-23 | Unsettled Liquidation Status

Category	Severity	Location	Status
Inconsistency	Minor	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 388; contracts/libraries/AuctionProxy.sol (pre): 72	Resolved

Description

The `AuctionProxy::buyFromAuction()` function is responsible for purchasing collateral from a liquidation auction. It calls the `ClipperLike::take()` function to execute the purchase. If the entire `lot` of collateral is purchased, the auction is removed:

```
412  // contracts/clip.sol
413  if (lot == 0) {
414      _remove(id);
415  } else if (tab == 0) {
```

However, if the auction is fully settled with no `leftover` collateral, the `PancakeSwapV3LpProvider::liquidation()` function is not called with `isLeftOver = true`. This is because the `leftover` variable in `AuctionProxy::buyFromAuction()` will be zero, and the call to `helioProvider.liquidation(..., true)` is skipped.

As a result, the user's liquidation record in `userLiquidations` is not cleared, and their `ongoing` status remains `true`. This prevents the user from performing any actions that require `userLiquidations[user].ongoing` to be false, such as providing more LPs.

Recommendation

A potential fix could be to check if the `lot` in the `clip.sales` mapping is zero after the `take` call. If it is, then the auction is finished, and `helioProvider.liquidation(..., true)` should be called.

Alleviation

[Lista Dao, 08/26/2025]: The team heeded the advice and resolved the issue by revising the liquidation flow and checking does the liquidation ended by the `lot` or `tab` value of user's `Sale` in commit 0510af0cc4157a6d7eb08d8b6e619e25a358dc13

LDA-25 | Approved Address Can Approve Other Addresses

Category	Severity	Location	Status
Design Issue	Minor	contracts/Interaction.sol (pre): 34; contracts/clip.sol (pre): 55; contracts/dog.sol (pre): 59; contracts/jar.sol (pre): 44; contracts/join.sol (pre): 71, 133; contracts/jug.sol (pre): 38; contracts/spot.sol (pre): 31; contracts/vat.sol (pre): 36; contracts/vow.sol (pre): 34	Resolved

Description

The `rely` function contains a permission-management issue. When an owner grants permissions or approval to another address (address B), this address B also gains the ability to approve other addresses with the same role. This can lead to unintended permission propagation and potential security risks.

Recommendation

Consider implementing a separate modifier to separate things apart.

Alleviation

[Lista Dao, 08/27/2025]: All "rely-ed" address are our own multi-sig wallets also they requires a 3/6 signatures threshold.

LDA-26 | Third-Party Dependency Usage

Category	Severity	Location	Status
Design Issue	Minor	oracle/PythOracle.sol (commit:50b1e7b): 4, 5; contracts/oracle/API3Oracle.sol (pre): 15; contracts/oracle/AsUsdfOracle.sol (pre): 9; contracts/oracle/BBtcOracle.sol (pre): 11; contracts/oracle/BtcOracle.sol (pre): 11; contracts/oracle/BusdOracle.sol (pre): 9; contracts/oracle/EthOracle.sol (pre): 12; contracts/oracle/EzEthOracle.sol (pre): 9; contracts/oracle/FdUsdOracle.sol (pre): 18; contracts/oracle/PumpBtcOracle.sol (pre): 9; contracts/oracle/PythOracle.sol (pre): 14; contracts/oracle/SlisBnbOracle.sol (pre): 19; contracts/oracle/SolvBTCBBNOracle.sol (pre): 20; contracts/oracle/SolvBtcOracle.sol (pre): 11; contracts/oracle/StoneOracle.sol (pre): 22; contracts/oracle/UsdfOracle.sol (pre): 18; contracts/oracle/UsdtOracle.sol (pre): 18; contracts/oracle/WBETHOracle.sol (pre): 9; contracts/oracle/asBnbOracle.sol (pre): 34; contracts/oracle/mBTCOracle.sol (pre): 9; contracts/oracle/mCAKEOracle.sol (pre): 9; contracts/oracle/mwBETHOracle.sol (pre): 10; contracts/oracle/priceFeeds/sUSDXPriceFeed.sol (pre): 18; contracts/oracle/sUsdxOracle.sol (pre): 9; contracts/oracle/wstETHOracle.sol (pre): 10	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Lista Dao, 08/27/2025]: Our internal security team has prepared a set or monitors, they a ready when the new contracts are deployed.

LDA-28 | Pyth Unchecked PublishTime

Category	Severity	Location	Status
Coding Issue	Minor	contracts/oracle/PythOracle.sol (pre): 18~21, 31~34, 36~54, 56~75	Resolved

Description

```
1   PythStructs.Price memory price = pyth.getPriceUnsafe(priceId);
```

This method returns the price object containing last updated price for the requested price feed ID.

This function may return a price from arbitrarily far in the past. It is the caller's responsibility to check the returned publishTime to ensure that the update is recent enough for their use case. If you need the latest price, update the price using updatePriceFeeds() and then call getPrice().

Reference: https://api-reference.pyth.network/price-feeds/evm/getPriceUnsafe

Recommendation

Check the publishTime of a Pyth price.

Alleviation

[Lista Dao, 08/27/2025]: This oracle is no-longer in use.

LDA-15 | Unused Function Parameters Can Lead To False Assumptions

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ceros/provider/BaseTokenProvider.sol (pre): 288	Acknowledged

Description

The _amount parameter is not used in the daoBurn function or just used in the emit event. Unused function parameters can mislead users, leading to false assumptions about the function's behaviour. This can cause confusion among developers, especially in event-driven systems where unused parameters in emitted events may propagate confusion downstream. To mitigate this, developers should either document the purpose of unused parameters or consider removing them altogether, ensuring clarity and maintainability in the codebase.

Recommendation

It's recommended to remove the unused parameters.

Alleviation

[Lista Dao, 08/27/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

LDA-16 | LP Price Sync In `PancakeSwapV3LpProvider::_deposit()`

Category	Severity	Location	Status
Inconsistency	Informational	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 588	Acknowledged

Description

The `PancakeSwapV3LpProvider::_deposit()` function calls `_syncUserCdpPosition(user, false)` with `syncLpPrice` set to `false`. If the user has previously provided LP tokens, their earlier deposits' value will not be synced within the CDP.

Although `syncUserLpValues()` and `batchSyncUserLpValues()` are functions designed for a bot to synchronize LP prices with the CDP, it relies on these external calls for CDP position correction.

Recommendation

Call `_syncUserCdpPosition(user, true)` inside `_deposit()` so that existing LP positions are priced at the latest market rates during each deposit.

Alleviation

[Lista Dao, 08/26/2025]: We would like to lower the gas consumption by not to refresh all LP values when user deposit a new LP.

Also, the off-chain bot will promptly call the `syncUserLpValues()` function if it's needed.

LDA-22 | Price Oracle Decimal Assumes A Fixed Value

Category	Severity	Location	Status
Inconsistency	Informational	contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol (pre): 71, 715	Acknowledged

Description

The `PancakeSwapV3LpProvider::getLpValue()` function is responsible for calculating the USD value of a PancakeSwap V3 LP token. It does this by getting the amounts of `token0` and `token1` in the LP position and multiplying them by their respective prices obtained from the `ResilientOracle`. The `ResilientOracle` in turn fetches prices from Chainlink price feeds.

The issue lies in the hardcoded `RESILIENT_ORACLE_DECIMALS` constant, which is set to `1e8`. The `getLpValue` function uses this constant to normalize the prices of `token0` and `token1`. However, Chainlink USD pairs do not have a standardized number of decimals. For example, the AMPL/USD feed uses 18 decimals. When the `ResilientOracle` returns a price with decimals different from 8, the `getLpValue` function will incorrectly calculate the value of the AMPL-X LP.

If the value of the collateral is inflated, users can borrow more LisUSD than they are entitled to, leading to bad debt for the protocol if the user defaults.

Recommendation

Instead of using a hardcoded decimal value, the `ResilientOracle` should be modified to expose the decimals of the underlying price feed. The `PancakeSwapV3LpProvider::getLpValue()` function should then use this information to correctly normalize the price.

Alleviation

[Lista Dao, 08/26/2025]: All price from the Resilient Oracle will return token price in 8 decimal places strictly.

LDA-29 Unchecked Arithmetic In Token Amount Normalization May Cause Overflow

Category	Severity	Location	Status
Volatile Code	Informational	ceros/provider/pancakeswapLpProvider/libraries/PcsV3LpNumbersHelper.sol (commit:50b1e7b): 63~73	Acknowledged

Description

The function getAmounts wraps arithmetic operations inside an unchecked block. This disables Solidity's default overflow and underflow protection, which can result in unexpected wrap-around behavior. Specifically, multiplications such as amount0 * (10 ** (18 - decimals0)) may overflow if amount0 is large, causing incorrect results. While current logic avoids negative exponentiation, the unchecked context introduces risk if token amounts increase beyond safe ranges.

Recommendation

Remove the unchecked block and rely on Solidity's default overflow checks.

Alleviation

[Lista Dao, 09/04/2025]: Issue acknowledged. I won't make any changes for the current version. As all V3 LP we use as collateral, the decimal places of the tokens will not larger than 18, if we really need to support token that has a higher decimal place, we will upgrade the library to address it.

FORMAL VERIFICATION | LISTA DAO - AUDIT 2

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of AccessControl-Enumerable v4.2

We verified properties of the public interface of contracts that provide an AccessControl-Enumerable-v4.2 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The functions `getRoleMember` and `getRoleMemberCount` retrieve an account with a specified `role` and count the total accounts with that `role`, respectively.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-grantrole-succeed-for-valid-inputs	`grantRole` Function Succeeds for Valid Inputs
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-grantrole-revert-no-admin	`grantRole` Reverts When Sender Is Not Admin
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role

Property Name	Title
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
erc165-supportsinterface-correct-false	`supportsInterface` Returns `False` for Id 0xffffffff
erc165-supportsinterface-succeed-always	`supportsInterface` Always Succeeds
erc165-supportsinterface-correct-erc165	`supportsInterface` Signals Support for ERC165
accesscontrolenumerable-getrolemember-succeed-for-valid-inputs	`getRoleMember` Succeeds for Valid Inputs
erc165-supportsinterface-no-change-state	`supportsInterface` Does Not Change the Contract's State
accesscontrolenumerable-getrolemembercount-change-state	`getRoleMemberCount` Changes No State Variables
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrolenumerable-getrolemember-change-state	`getRoleMember` Changes No State Variables
accesscontrol-renouncerole-succeed-for-valid-inputs	`renounceRole` Function Succeeds for Valid Inputs
accesscontrol-revokerole-succeed-for-valid-inputs	`revokeRole` Function Succeeds for Valid Inputs
accesscontrolenumerable-renouncerole-not-member-already	`renounceRole` Does Not Remove Non-Member
accesscontrolenumerable-renouncerole-remove-member	`renounceRole` Removes Member from Role
accesscontrolenumerable-revokerole-not-member-already	`revokeRole` Does Not Remove Non-Member
accesscontrolenumerable-revokerole-remove-member	`revokeRole` Removes Member from Role

Property Name	Title
accesscontrolenumerable-grantRole-member-already	`grantRole` Does Not Add Member Already in Role
accesscontrolenumerable-grantRole-add-member	`grantRole` Adds Member to Role
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address
accesscontrolenumerable-getrolemembercount-succeed-always	`getRoleMemberCount` Always Succeeds
accesscontrol-supportsinterface-correct-accesscontrol	`supportsInterface` Signals that AccessControl is Implemented
accesscontrolenumerable-supportsinterface-correct-accesscontrolenumerable	`supportsInterface` Signals Support for `AccessControlEnumerable`
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-revokerole-revert-no-admin	`revokeRole` Reverts When Sender Is Not Admin

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
ownable-owner-succeed-normal	`owner` Always Succeeds
ownable-renounceownership-correct	Ownership is Removed

Property Name	Title
ownable-transferownership-correct	Ownership is Transferred
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and

- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address

Property Name	Title
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract AsBnbOracle (contracts/oracle/asBnbOracle.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results For Contract LpUsd (contracts/ceros/provider/LpUsd.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of ERC-20 Compliance

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-false	True	
erc20-approve-never-return-false	True	
erc20-approve-succeed-normal	True	
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-change-state	True	
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-revert-zero	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-correct-amount	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-false	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-allowance	True	

Detailed Results For Contract PumpBTCProvider (contracts/ceros/provider/PumpBTCProvider.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	True	
accesscontrol-renouncerole-revert-not-sender	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	True	
accesscontrol-getroleadmin-succeed-always	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	True	
accesscontrol-hasrole-succeed-always	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results For Contract mBTCProvider (contracts/ceros/provider/mBTCProvider.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results For Contract Interaction (contracts/Interaction.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract PancakeSwapV3LpStakingVault (contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingVault.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of AccessControl-Enumerable v4.2

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-succeed-for-valid-inputs	True	
accesscontrol-grantrole-revert-no-admin	True	
accesscontrol-grantrole-correct-role-granting	True	
accesscontrolenumerable-grantRole-member-already	Inconclusive	
accesscontrolenumerable-grantRole-add-member	Inconclusive	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	True	
accesscontrol-getroleadmin-succeed-always	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	
accesscontrol-revokerole-succeed-for-valid-inputs	Inconclusive	
accesscontrolenumerable-revokerole-not-member-already	Inconclusive	
accesscontrolenumerable-revokerole-remove-member	Inconclusive	
accesscontrol-revokerole-revert-no-admin	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `supportsInterface`

Property Name	Final Result	Remarks
erc165-supportsinterface-correct-false	True	
erc165-supportsinterface-succeed-always	True	
erc165-supportsinterface-correct-erc165	True	
erc165-supportsinterface-no-change-state	True	
accesscontrol-supportsinterface-correct-accesscontrol	True	
accesscontrolenumerable-supportsinterface-correct-accesscontrolenumerable	True	

Detailed Results for Function `getRoleMember`

Property Name	Final Result	Remarks
accesscontrolenumerable-getrolemember-succeed-for-valid-inputs	True	
accesscontrolenumerable-getrolemember-change-state	True	

Detailed Results for Function `getRoleMemberCount`

Property Name	Final Result	Remarks
accesscontrolenumerable-getrolemembercount-change-state	True	
accesscontrolenumerable-getrolemembercount-succeed-always	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-for-valid-inputs	Inconclusive	
accesscontrolenumerable-renouncerole-not-member-already	Inconclusive	
accesscontrolenumerable-renouncerole-remove-member	Inconclusive	
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results For Contract HelioOracle (contracts/oracle/HelioOracle.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of Standard Ownable Properties

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inapplicable	The property does not apply to the contract

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results For Contract ResilientOracle (contracts/oracle/ResilientOracle.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inconclusive	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results For Contract PancakeSwapV3LpStakingHub (contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of AccessControl-Enumerable v4.2

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	
accesscontrol-renouncerole-succeed-for-valid-inputs	Inconclusive	
accesscontrolenumerable-renouncerole-not-member-already	Inconclusive	
accesscontrolenumerable-renouncerole-remove-member	Inconclusive	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `supportsInterface`

Property Name	Final Result	Remarks
erc165-supportsinterface-succeed-always	True	
erc165-supportsinterface-correct-erc165	True	
erc165-supportsinterface-correct-false	True	
accesscontrol-supportsinterface-correct-accesscontrol	True	
accesscontrolenumerable-supportsinterface-correct-accesscontrolenumerable	True	
erc165-supportsinterface-no-change-state	True	

Detailed Results for Function `getRoleMemberCount`

Property Name	Final Result	Remarks
accesscontrolenumerable-getrolemembercount-succeed-always	True	
accesscontrolenumerable-getrolemembercount-change-state	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-succeed-for-valid-inputs	True	
accesscontrol-grantrole-revert-no-admin	True	
accesscontrol-grantrole-correct-role-granting	True	
accesscontrolenumerable-grantRole-member-already	Inconclusive	
accesscontrolenumerable-grantRole-add-member	Inconclusive	

Detailed Results for Function `getRoleMember`

Property Name	Final Result	Remarks
accesscontrolenumerable-getrolemember-succeed-for-valid-inputs	True	
accesscontrolenumerable-getrolemember-change-state	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	
accesscontrol-revokerole-revert-no-admin	True	
accesscontrol-revokerole-succeed-for-valid-inputs	Inconclusive	
accesscontrolenumerable-revokerole-not-member-already	Inconclusive	
accesscontrolenumerable-revokerole-remove-member	Inconclusive	

Detailed Results For Contract LisUSD (contracts/LisUSD.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of ERC-20 Compliance

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-never-return-false	True	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-false	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-correct-value	True	
erc20-balanceof-succeed-always	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-allowance	Inapplicable	The property does not apply to the contract
erc20-transferfrom-correct-amount	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	True	
erc20-transfer-never-return-false	True	
erc20-transfer-false	True	
erc20-transfer-revert-zero	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-correct-amount	True	

Detailed Results For Contract Vat (contracts/vat.sol) In Commit d603a5c208cef3d0f49ce6896300701f7c5b27fe

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inapplicable	The property does not apply to the contract

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

APPENDIX | LISTA DAO - AUDIT 2

Audit Scope

lista-dao/lista-dao-contracts
contracts/Interaction.sol
contracts/libraries/AuctionProxy.sol
contracts/ceros/provider/BaseTokenProvider.sol
contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpProvider.sol
contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingVault.sol
contracts/oracle/API3Oracle.sol
contracts/oracle/AsUsdfOracle.sol
contracts/oracle/BBtcOracle.sol
contracts/oracle/BtcOracle.sol
contracts/oracle/BusdOracle.sol
contracts/oracle/EthOracle.sol
contracts/oracle/EzEthOracle.sol
contracts/oracle/FdUsdOracle.sol
contracts/oracle/PumpBtcOracle.sol
contracts/oracle/PythOracle.sol
contracts/oracle/SlisBnbOracle.sol
contracts/oracle/SolvBTCBBNOracle.sol
contracts/oracle/SolvBtcOracle.sol
contracts/oracle/StoneOracle.sol

lista-dao/lista-dao-contracts
contracts/oracle/UsdfOracle.sol
contracts/oracle/UsdtOracle.sol
contracts/oracle/WBETHOracle.sol
contracts/oracle/asBnbOracle.sol
contracts/oracle/mBTCOracle.sol
contracts/oracle/mCAKEOracle.sol
contracts/oracle/mwBETHOracle.sol
contracts/oracle/sUsdxOracle.sol
contracts/oracle/wstETHOracle.sol
contracts/vow.sol
contracts/jug.sol
contracts/join.sol
contracts/jar.sol
contracts/dog.sol
contracts/clip.sol
contracts/spot.sol
contracts/vat.sol
contracts/ceros/provider/mBTCProvider.sol
contracts/ceros/provider/PumpBTCProvider.sol
contracts/ceros/provider/pancakeswapLpProvider/PancakeSwapV3LpStakingHub.sol
contracts/LisUSD.sol
contracts/ceros/provider/LpUsd.sol

lista-dao/lista-dao-contracts
contracts/ceros/provider/pancakeswapLpProvider/libraries/PcsV3LpLiquidationHelper.sol
contracts/ceros/provider/pancakeswapLpProvider/libraries/PcsV3LpNumbersHelper.sol
contracts/amo/DynamicDutyCalculator.sol
contracts/oracle/BnbOracle.sol
contracts/oracle/HelioOracle.sol
contracts/oracle/ResilientOracle.sol
contracts/oracle/Usd1Oracle.sol
contracts/oracle/WeEthOracle.sol

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the

semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-Enumerable-v4.2 Properties

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

accesscontrol-grantrole-revert-no-admin

The `grantRole` function must revert if the sender does not have the appropriate admin role.

Specification:

```
reverts_when !hasRole(getRoleAdmin(role), msg.sender);
```

accesscontrol-grantrole-succeed-for-valid-inputs

The `grantRole` function must succeed when the sender has the appropriate admin role.

Specification:

```
requires hasRole(getRoleAdmin(role), msg.sender);
reverts_only_when false;
also
ensures true;;
```

accesscontrolenumerable-grantRole-add-member

The `grantRole` function in contract PancakeSwapV3LpStakingVault must add a member to the specified role.

Specification:

```
requires !hasRole(role, account);
ensures \old(getRoleMemberCount(role)) + 1 == getRoleMemberCount(role);
ensures getRoleMember(role, getRoleMemberCount(role) - 1) == account;
also
ensures true;
```

accesscontrolenumerable-grantRole-add-member

The `grantRole` function in contract PancakeSwapV3LpStakingHub must add a member to the specified role.

Specification:

```
requires !hasRole(role, account);
ensures \old(getRoleMemberCount(role)) + 1 == getRoleMemberCount(role);
ensures getRoleMember(role, getRoleMemberCount(role) - 1) == account;
also
ensures true;
```

accesscontrolenumerable-grantRole-member-already

The `grantRole` function in contract PancakeSwapV3LpStakingVault must not add a member to the role if the member is already present.

Specification:

```
requires hasRole(role, account);
ensures \old(getRoleMemberCount(role)) == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-grantRole-member-already

The `grantRole` function in contract PancakeSwapV3LpStakingHub must not add a member to the role if the member is already present.

Specification:

```
requires hasRole(role, account);
ensures \old(getRoleMemberCount(role)) == getRoleMemberCount(role);
also
ensures true;
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

accesscontrol-revokerole-revert-no-admin

The `revokeRole` function must revert if the sender does not have the appropriate admin role.

Specification:

```
reverts_when !hasRole(getRoleAdmin(role), msg.sender);
```

accesscontrol-revokerole-succeed-for-valid-inputs

The `revokeRole` function must succeed when the sender has the appropriate admin role.

Specification:

```
requires hasRole(getRoleAdmin(role), msg.sender);
reverts_only_when false;
also
ensures true;
```

accesscontrolenumerable-revokerole-not-member-already

The `revokeRole` function in contract PancakeSwapV3LpStakingVault must not remove a member from the role if the member is not present.

Specification:

```
requires !hasRole(role, account);
ensures \old(getRoleMemberCount(role)) == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-revokerole-not-member-already

The `revokeRole` function in contract PancakeSwapV3LpStakingHub must not remove a member from the role if the member is not present.

Specification:

```
requires !hasRole(role, account);
ensures \old(getRoleMemberCount(role)) == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-revokerole-remove-member

The `revokeRole` function in contract PancakeSwapV3LpStakingVault must remove a member from the specified role.

Specification:

```
requires hasRole(role, account);
ensures \old(getRoleMemberCount(role)) - 1 == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-revokerole-remove-member

The `revokeRole` function in contract PancakeSwapV3LpStakingHub must remove a member from the specified role.

Specification:

```
requires hasRole(role, account);
ensures \old(getRoleMemberCount(role)) - 1 == getRoleMemberCount(role);
also
ensures true;
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `supportsInterface`

accesscontrol-supportsinterface-correct-accesscontrol

A call of `supportsInterface(interfaceId)` with the interface id of AccessControl must return true.

Specification:

```
requires interfaceId == 0x7965db0b;;
ensures \result;
```

accesscontrolenumerable-supportsinterface-correct-accesscontrolenumerable

Invocations of `supportsInterface(id)` must signal that the interface `AccessControlEnumerable` is implemented.

Specification:

```
requires interfaceId == 0x5a05180f;
ensures \result;
```

erc165-supportsinterface-correct-erc165

Invocations of `supportsInterface(id)` must signal that the interface `ERC165` is implemented.

Specification:

```
requires interfaceId == 0x01ffc9a7;
ensures \result;
```

erc165-supportsinterface-correct-false

Invocations of `supportsInterface(id)` with `id` 0xffffffff must return `false`.

Specification:

```
requires interfaceId == 0xffffffff;
ensures !\result;
```

erc165-supportsinterface-no-change-state

Function `supportsInterface` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc165-supportsinterface-succeed-always

Function `supportsInterface` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `getRoleMember`

accesscontrolenumerable-getrolemember-change-state

The `getRoleMember` function in contract PancakeSwapV3LpStakingVault must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrolenumerable-getrolemember-change-state

The `getRoleMember` function in contract PancakeSwapV3LpStakingHub must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrolenumerable-getrolemember-succeed-for-valid-inputs

The `getRoleMember` function in contract PancakeSwapV3LpStakingVault must succeed when provided with valid inputs.

Specification:

```
requires index < getRoleMemberCount(role);
reverts_only_when false;
```

accesscontrolenumerable-getrolemember-succeed-for-valid-inputs

The `getRoleMember` function in contract PancakeSwapV3LpStakingHub must succeed when provided with valid inputs.

Specification:

```
requires index < getRoleMemberCount(role);
reverts_only_when false;
```

Properties related to function `getRoleMemberCount`

accesscontrolenumerable-getrolemembercount-change-state

The `getRoleMemberCount` function in contract PancakeSwapV3LpStakingVault must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrolenumerable-getrolemembercount-change-state

The `getRoleMemberCount` function in contract PancakeSwapV3LpStakingHub must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrolenumerable-getrolemembercount-succeed-always

The `getRoleMemberCount` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-for-valid-inputs

The `renounceRole` function must succeed when the caller is the same as the `account`.

Specification:

```
requires account == msg.sender;
reverts_only_when false;
also
ensures true;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

accesscontrolenumerable-renouncerole-not-member-already

The `renounceRole` function in contract PancakeSwapV3LpStakingVault must not remove a member from the role if the member is not present.

Specification:

```
requires !hasRole(role, account);
ensures \old(getRoleMemberCount(role)) == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-renouncerole-not-member-already

The `renounceRole` function in contract PancakeSwapV3LpStakingHub must not remove a member from the role if the member is not present.

Specification:

```
requires !hasRole(role, account);
ensures \old(getRoleMemberCount(role)) == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-renouncerole-remove-member

The `renounceRole` function in contract PancakeSwapV3LpStakingVault must remove a member from the specified role.

Specification:

```
requires hasRole(role, account);
ensures \old(getRoleMemberCount(role)) - 1 == getRoleMemberCount(role);
also
ensures true;
```

accesscontrolenumerable-renouncerole-remove-member

The `renounceRole` function in contract PancakeSwapV3LpStakingHub must remove a member from the specified role.

Specification:

```
requires hasRole(role, account);
ensures \old(getRoleMemberCount(role)) - 1 == getRoleMemberCount(role);
also
ensures true;
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Description of the Analyzed ERC-20 Properties

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract LpUsd must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-change-state

The `totalSupply` function in contract LisUSD must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract LpUsd.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract LisUSD.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `LpUsd` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-false

If the `transfer` function in contract `LisUSD` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                  && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

Description of the Analyzed Ownable Properties

Properties related to function `owner`

ownable-owner-succeed-normal

Function `owner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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 CertiK 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 CertiK 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.