Accretion

SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for Hylo
July 08, 2026
A26HYL1

Hylo

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Lead Auditor	Mahdi Rostami (mahdi@accretion.xyz)

CLIENT

Hylo (https://hylo.so) engaged Accretion to conduct a security assessment of the second version of the Hylo protocol on Solana which implements exo pairs, a new earn pool, and a revised rebalancing mechanism.

ENGAGEMENT TIMELINE

- 23 Feb — Project Kickoff
  Initial planning and scope definition
- 23 Feb — Assessment Begins
  Security review and testing phase
- 07 Apr — Review Fixes
  Security recommendations are given and implemented
- 02 Jul — Project Completion
  Report delivery and on-chain confirmation

AUDITED CODE

#	Program ID	Repository
1	HysTabVUfmQBFcmzu1ctRd1Y1fxd66RBpboy1bmtDSQQ	https://github.com/hylo-so/protocol
2	HYEXCHtHkBagdStcJCp3xbbb9B7sdMdWXFNj6mdsG4hn	https://github.com/hylo-so/protocol

ASSESSMENT

The security assessment of Hylo's Protocol V2 revealed 51 findings across all severity levels: 16 medium, 25 low, and 10 informational; no critical or high severity issues were identified. Of these, 44 were fixed, 6 accepted or acknowledged. Medium findings centered on fee miscalculations, oracle staleness, depeg oscillation, and LP mispricing. The absence of critical or high issues reflects a reasonably secure protocol foundation.

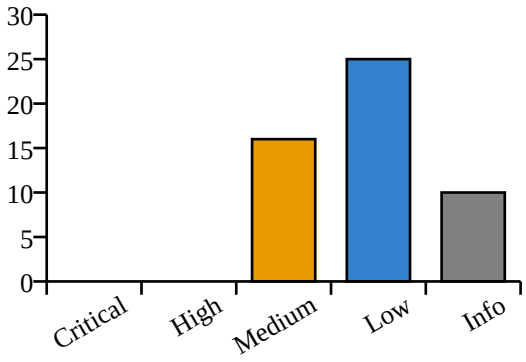
CODE ASSESSMENT

The codebase uses Anchor on Solana with structured PDA derivation and event emission. Security practices are generally sound, though several issues indicate gaps in input validation, idempotency guards, and invariant checks across composable instruction paths.

KEY FINDINGS

No critical or high severity issues were found. The most impactful medium findings include mint fee overcharging, LP drawdown debt mispricing, borrow-rate harvest skipping elapsed epochs, rebalance loss underaccounting, and first-minter NAV inflation on exo pairs.

SEVERITY DISTRIBUTION



ENGAGEMENT SCOPE

The scope of this security assessment was a full review of the following items:

Item 1: Earn Pool

Link: https://github.com/hylo-so/protocol
Commit: init-v2
Program ID: HysTabVUfmQBFcmzu1ctRd1Y1fxd66RBpboy1bmtDSQQ
Audit Result:
- **Audited Commit:** bd95c08e6ce40bbcf8a315895dc193ff56ce851e
- **Build Hash:** bbcdc2555ca929842ae22060b93962615d8a5a849a57d16e8f3a65c0405965d2
- **Status:** unverified
- **Comment:** not verified

Item 2: Exchange

Link: https://github.com/hylo-so/protocol
Commit: init-v2
Program ID: HYEXCHtHkBagdStcJCp3xbbb9B7sdMdWXFNj6mdsG4hn
Audit Result:
- **Audited Commit:** bd95c08e6ce40bbcf8a315895dc193ff56ce851e
- **Build Hash:** 958348f892e30a529fd9ac9a240afa516f8d55f90e428d63227903fde8bdf577
- **Status:** unverified
- **Comment:** not verified

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-M1	LST-USDC Swaps Ignore SPL Stake Pool Withdrawal Fees and Misprice Rebalance Trades	MEDIUM	ACKNOWLEDGED
ACC-M2	Borrow-Rate Harvest Cap Can Overmint HYUSD and Push CR Below the Boundary	MEDIUM	FIXED
ACC-M3	Treating Zero Stablecoin Supply as Maximum CR Creates Dangerous Economic Edge Cases	MEDIUM	FIXED
ACC-M4	New Exo Pair Levercoin NAV Inflation Can Steal Value From Later Minters	MEDIUM	FIXED
ACC-M5	Rebalance Loss Underaccounting Can Leave the Protocol Undercollateralized When the Earn Pool Is Short	MEDIUM	FIXED
ACC-M6	Stablecoin Profit Mint Can Push Rebalance Settlement Into Depeg After the Check	MEDIUM	FIXED
ACC-M7	Stale Collateral Vault Balance Can Bypass Depeg Check During Exo PnL Settlement	MEDIUM	FIXED
ACC-M8	Marinade-Family LSTs Can Be Mispriced in Rebalance Swaps	MEDIUM	FIXED
ACC-M9	Missed Borrow-Rate Harvests Are Not Accrued Across Elapsed Epochs	MEDIUM	FIXED
ACC-M10	Address Update Process Can Accept Malicious Admin Address Using Bait And Switch	MEDIUM	FIXED
ACC-M11	Exact Drawdown Repayments Can Cause Depeg Oscillation	MEDIUM	FIXED
ACC-M12	Missing Harvest Gates Can Double-Count Drawdown Repayments	MEDIUM	FIXED
ACC-M13	LP Accounting Ignores Drawdown Debt and Misprices sHYUSD	MEDIUM	ACKNOWLEDGED
ACC-M14	Mint Fee Uses Pre-Fee Amount and Overcharges Users	MEDIUM	ACKNOWLEDGED
ACC-M15	Mint Fee Reverts in Valid Mode1 Range (CR 130–150)	MEDIUM	FIXED
ACC-M16	Using init for ATA Creation Enables Pre-Init DoS	MEDIUM	FIXED
ACC-L1	Oracle Freeze Locks out Settlement While Allowing Users To Exit Earn Pool	LOW	FIXED
ACC-L2	USDC Sell Swaps Ignore Drawdown and Can Sell Collateral While the Pair Is Effectively Depegged	LOW	FIXED
ACC-L3	Stablecoin Minting Ignores Drawdown and Can Overstate Max Mintable Amount	LOW	FIXED
ACC-L4	Zero Free Collateral Can Make Levercoin NAV Zero and Panic Levercoin Minting	LOW	FIXED

ACC-L5	LST-to-LST Swaps Can Leave Total SOL Cache Out of Sync	LOW	FIXED
ACC-L6	Zero Levercoin Supply Can Reset NAV and Misprice Future Levercoin Minters	LOW	FIXED
ACC-L7	Hardcoded 1-Second Oracle Window Can Break Rebalance Swaps	LOW	FIXED
ACC-L8	No-op LST Harvest Leaves Per-LST Harvest Cursors Stale	LOW	FIXED
ACC-L9	Dust xSOL Can Permanently Block Levercoin Pool Deprecation	LOW	FIXED
ACC-L10	Admin Transfer Can Brick Payer-Based Admin Instructions	LOW	FIXED
ACC-L11	Active Stable-to-Lever Path During Upgrade Can Leave Earn Pool With xSOL and Overmint sHYUSD	LOW	FIXED
ACC-L12	Slippage Is Silently Ignored for sHYUSD Deposit and Withdraw Routes	LOW	FIXED
ACC-L13	Collapsed SwapLstToLst Route Can Execute the Swap in the Wrong Direction	LOW	FIXED
ACC-L14	Borrow Rate Cap Can Double After Reduced Solana Epoch Duration	LOW	FIXED
ACC-L15	Wrong Levercoin Mint PDA Prevents Pool Deprecation	LOW	FIXED
ACC-L16	Registering a new LST before harvest_yield blocks yield harvest for the epoch	LOW	FIXED
ACC-L17	Missing Minimum-Output Check Can Cause Silent Donation	LOW	FIXED
ACC-L18	initialize_lst_registry_calculators is not idempotent and can corrupt the registry LUT on repeated calls	LOW	FIXED
ACC-L19	Redeem Event Emits Incorrect Price Field	LOW	FIXED
ACC-L20	Stale Oracle Data Can Freeze Stability Pool Withdrawals	LOW	FIXED
ACC-L21	Stability pool PoolConfig setters allow no-op updates	LOW	FIXED
ACC-L22	Hylo LST swap fee update stores unvalidated values	LOW	FIXED
ACC-L23	create_metadata Incorrectly Passes Mint as Rent and System Program	LOW	FIXED
ACC-L24	Shared PDA Seeds Can Collide and Merge Vault Control Across Pairs	LOW	FIXED
ACC-L25	Front-Running Can Desync HYLO Virtual Stablecoin From HYUSD Supply	LOW	ACKNOWLEDGED
ACC-I1	Same-Epoch Stale LST Prices Can Misprice LST Swaps and Leak Vault Value	INFO	ACKNOWLEDGED
ACC-I2	Repeated Same-Epoch LST Price Updates Can Reduce Harvested Yield	INFO	FIXED

ACC-I3	Zero Earn Pool Balance Can Block New Deposits While LP Supply Remains	INFO	FIXED
ACC-I4	100% Borrow Rate Fee Can Redirect All Exo Borrow Harvest Away From the Earn Pool	INFO	FIXED
ACC-I5	Misleading Self-CPI Comment Can Lead to Incorrect Rebalance Settlement Assumptions	INFO	FIXED
ACC-I6	Unused Earn-Pool Error Codes	INFO	FIXED
ACC-I7	First Depositor Can Inflate sHYUSD Share Price and Steal Later Deposits	INFO	FIXED
ACC-I8	Hardcoded 9-Decimal Assumption Can Misprice Non-9-Decimal LSTs	INFO	ACKNOWLEDGED
ACC-I9	UpdateLstPricesEvent Can Overflow When More Than 10 LSTs Are Registered	INFO	ACKNOWLEDGED
ACC-I10	New LP NAV Ignores Existing xSOL, Letting New Depositors Drain Pool	INFO	FIXED

DETAILED ISSUES

ACC-M1 LST-USDC Swaps Ignore SPL Stake Pool Withdrawal Fees and Misprice Rebalance Trades

MEDIUM ACKNOWLEDGED

Description

`swap_lst_usdc` calculates the LST true price from the stake pool's `total_lamports / pool_token_supply`. This ignores the SPL stake pool withdrawal fee.

The Sanctum SPL calculator includes the stake withdrawal fee when converting LST to SOL. It first applies the withdrawal fee to the pool tokens, then converts the remaining amount to lamports. See calc.rs.

Because Hylo ignores this fee, the swap path can overstate the effective SOL value of the LST. This can make LST-to-USDC swaps overpay USDC for LST collateral and make the reverse direction use a price that does not match the value the protocol can actually realize.

Location

- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L239-L245
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L424-L430
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/lst/stake_pool.rs#L11-L24
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/lst/stake_pool.rs#L31-L70
- https://github.com/igneous-labs/S/blob/master/libs/sol-value-calculator-programs/spl-calculator-lib/src/calc.rs#L92

Relevant Code

```
// True LST price from stake pool
let stake_pool = SplStakePool::from_bytes(&pool_state.data.borrow())?;
let true_price = stake_pool.true_price()?;

// Discount true price by rebalancing fee
let adjusted_lst_sol_price =
  true_price.adjust_price(lst_header.rebalance_fee()?)?;
```

```
pub struct SplStakePool {
  pub total_lamports: UFix64<N9>,
  pub pool_token_supply: UFix64<N9>,
  pub last_update_epoch: u64,
}
```

```
pub fn true_price(&self) -> Result<LstSolPrice> {
  let price = self
    .total_lamports
    .mul_div_floor(UFix64::one(), self.pool_token_supply)
    .ok_or(CoreError::StakePoolDivByZero)?;
  Ok(LstSolPrice::new(price.into(), self.last_update_epoch))
}
```

```
let aaf = self.stake_withdrawal_fee()?.apply(pool_tokens)?;
let pool_tokens_burnt = aaf.amt_after_fee();
let withdraw_lamports = self.lst_to_lamports_ratio().apply(pool_tokens_burnt)?;
```

Mitigation Suggestion

Use the Sanctum SPL calculator logic for SPL-family LST true price calculation, including the stake withdrawal fee.

The price used by `swap_lst_usdc` should reflect the effective SOL value after the stake pool withdrawal fee, not only `total_lamports / pool_token_supply`.

Remediation

Acknowledged.

ACC-M2 Borrow-Rate Harvest Cap Can Overmint HYUSD and Push CR Below the Boundary

MEDIUM FIXED

Description

`commit_normal_harvest` in `harvest_borrow_rate` caps the harvested HYUSD with `validate_stablecoin_amount()`. That function uses `max_mintable_stablecoin()`, which is the cap for collateral-backed stablecoin minting.

That formula assumes the mint is paired with a collateral deposit, so both TVL and stablecoin supply increase. Borrow-rate harvest does not add collateral. It only mints HYUSD from the borrow obligation, so TVL stays flat while stablecoin supply increases.

The correct cap for this TVL-neutral case is `TVL / target - supply`, which already exists as `max_swappable_stablecoin()`. Using the deposit formula makes the cap about `3.86x` too large at the `1.35` target CR, so a capped harvest can still push the system below the intended boundary into SellZone or Depeg.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L212-L221

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L232-L233

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/mod.rs#L293-L355

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_math.rs#L53-L103

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/rebalance/mode.rs#L83-L89

Relevant Code

```
let stablecoin_to_harvest = {
  let raw_stablecoin = exo_pair
    .borrow_rate_config
    .apply_borrow_rate(levercoin_market_cap)?
    .checked_convert()
    .ok_or(ErrorCode::TokenAmountPrecisionError)?;

  exchange_context
    .validate_stablecoin_amount(raw_stablecoin)
    .or_else(|_| exchange_context.max_mintable_stablecoin())?
};
```

```
exo_pair.virtual_stablecoin.mint(stablecoin_to_harvest)?;
```

```
fn validate_stablecoin_amount(
  &self,
  requested: UFix64<N6>,
) -> Result<UFix64<N6>> {
  let max = self.max_mintable_stablecoin()?;
```

```
/// max_stablecoin = (tvl - target_cr * cur_stablecoin) / (target_cr - 1)
pub fn max_mintable_stablecoin(
```

```
/// max_swappable = total_value_locked / target_collateral_ratio - stablecoin_supply
pub fn max_swappable_stablecoin(
```

Mitigation Suggestion

Use the TVL-neutral cap for borrow-rate harvest. This path should clamp against `max_swappable_stablecoin()` or a new explicit harvest cap that computes:

```
max_harvest = TVL / target_cr - stablecoin_supply
```

Do not use the collateral-deposit mint cap when no collateral is added.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/eebe7d2cc90c77f72c5a85ecf8a07fdb62e065b4.

ACC-M3 Treating Zero Stablecoin Supply as Maximum CR Creates Dangerous Economic Edge Cases

MEDIUM FIXED

Description

We found that the current implementation treats the collateral ratio (CR) as maximum whenever the virtual stablecoin supply is zero. This is problematic because a maximum CR effectively places the pair in the deepest “buy” state regardless of the actual collateral composition or economic reality of the pool.

Moreover, reaching this state is not necessarily difficult. Users can potentially redeem all virtual stablecoins from a pair, causing the CR calculation to jump to its maximum value. While some fee mechanisms exist to discourage this, the introduction of multiple EXO pairs with independent CRs creates opportunities to route around those fees.

For example, an attacker could:

1. Wait for a new pair. 2. Mint 1 levercoin. 3. Transfer collateral to it to bypass the fee. (Instead of going through the minting path, we use this path for 0 fee). 4. Swap EXO to USDC, gaining some profit and hurting the earn pool, with no fee. 5. Find a high-CR pair and mint a stablecoin with no fee. 6. Redeem the stablecoin here, paying only some fee. 7. Repeat this again cause CR is at the max again. 8. Note that the attacker owns all the collateral because there is only 1 lever coin.

Additionally, newly created pairs naturally start in this maximum-CR state if no seed minting is performed, providing an immediate source of economic imbalance.

As a result, the protocol may expose unintended fee bypasses, cross-pair arbitrage opportunities, and value extraction paths that can negatively impact liquidity providers and earn pools.

Location

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_math.rs#L14-L25

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/mod.rs#L58-L66

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/mod.rs#L176-L188

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/exo.rs#L124-L131

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/exo.rs#L156-L177

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/exo.rs#L184-L206

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_stablecoin_exo.rs#L139-L188

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/redeem_stablecoin_exo.rs#L129-L167

Relevant Code

```
pub fn collateral_ratio(
  total_collateral: UFix64<N9>,
  usd_collateral_price: UFix64<N9>,
  amount_stablecoin: UFix64<N6>,
) -> Result<UFix64<N9>> {
  if amount_stablecoin == UFix64::zero() {
    Ok(UFix64::new(u64::MAX))
  } else {
```

```
    amount_stablecoin
      .checked_convert::<N9>()
      .and_then(|stablecoin| {
        total_collateral.mul_div_floor(usd_collateral_price, stablecoin)



fn stablecoin_mint_enabled(&self) -> bool {
  self.collateral_ratio() >= self.stablecoin_mint_threshold()
}

fn levercoin_mint_enabled(&self) -> bool {
  self.rebalance_mode() != RebalanceMode::Depeg
}



fn stablecoin_nav(&self) -> Result<UFix64<N9>> {
  match self.rebalance_mode() {
    RebalanceMode::Depeg => depeg_stablecoin_nav(
      self.total_collateral(),
      self.collateral_usd_price().lower,
      self.virtual_stablecoin_supply()?,
    ),
    _ => Ok(UFix64::one()),
  }



let stablecoin_supply = virtual_stablecoin.supply()?;
let collateral_ratio = collateral_ratio(
  total_collateral,
  collateral_usd_price.lower,
  stablecoin_supply,
)?;
let rebalance_mode = RebalanceMode::from_cr(collateral_ratio);



let new_stablecoin = self
  .virtual_stablecoin_supply()?
  .checked_sub(&stablecoin_redeemed)
  .ok_or(DestinationStablecoin)?;
let projected_cr = collateral_ratio(
  new_total,
  self.collateral_usd_price.lower,
  new_stablecoin,
)?;



require!(
  exchange_context.stablecoin_mint_enabled(),
  ErrorCode::StablecoinMintDisabled,
);

// ...

exo_pair.virtual_stablecoin.mint(stablecoin_to_mint)?;



let FeeExtract {
  fees_extracted,
  amount_remaining,
} = exchange_context.stablecoin_redeem_fee(amount_collateral_out)?;

// ...

exo_pair.virtual_stablecoin.burn(stablecoin_to_redeem)?;
```

Mitigation Suggestion

Handle the zero-supply case differently instead of treating it as maximum CR. For example, use a capped CR value, a dedicated bootstrap state, or require minimum liquidity before normal CR-based pricing becomes active.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/0599e8fba1d2d8627a39652a713667333d1eecf8, https://github.com/hylo-so/sdk/commit/eb0f2034d430b6efb5ab9f146751f7d52fbf5a11, and https://github.com/hylo-so/protocol/commit/b0df4a6fdf1a444661fed66cb38b52cdebbe2624.

ACC-M4 New Exo Pair Levercoin NAV Inflation Can Steal Value From Later Minters

MEDIUM FIXED

Description

New Exo pairs are vulnerable to a first-minter share price inflation attack.

Levercoin NAV is calculated from the collateral vault balance and the levercoin supply. An attacker can mint a very small amount of levercoin, then transfer collateral directly into the pair's collateral vault. This inflates the levercoin NAV without minting more levercoin.

A later user who mints levercoin receives fewer tokens because the NAV is inflated and the conversion rounds down. The attacker can then redeem their levercoin at the inflated NAV and capture part of the victim's deposit.

The slippage check can reduce this attack vector when users set a strict minimum output, but it does not fully remove the issue for users or integrations that use loose slippage, omit slippage, or compute expectations from an already-inflated NAV.

This is most practical on new Exo pairs where levercoin supply is very small.

Location

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_levercoin_exo.rs#L137-L180

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_levercoin_exo.rs#L199-L221

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/redeem_levercoin_exo.rs#L130-L165

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/redeem_levercoin_exo.rs#L177-L214

Relevant Code

```
let collateral_vault_amount =
  normalize_mint_exp(collateral_mint, collateral_vault.amount)?;
let exchange_context = ExoExchangeContext::load(
  Clock::get()?,
  collateral_vault_amount,
  exo_pair.stablecoin_mint_threshold()?,
  exo_pair.oracle_config()?,
  exo_pair.levercoin_fees,
  collateral_usd_pyth_feed,
  exo_pair.virtual_stablecoin,
  Some(levercoin_mint),
  exo_pair.rebalance_deviation_tolerance()?,
  exo_pair.sell_curve_config,
  exo_pair.buy_curve_config,
  exo_pair.levercoin_market_cap_limit()?,
)?;

let levercoin_nav = exchange_context.levercoin_mint_nav()?;
let levercoin_to_mint = {
  let proposed_mint = exchange_context
    .exo_conversion()
    .exo_to_token(amount_remaining, levercoin_nav)?;
```

```
// Deposit collateral to vault
deposit_collateral(
  user_collateral_ta,
```

```
  collateral_vault,
  collateral_mint,
  user,
  token_program,
  denormalize_mint_exp(collateral_mint, amount_remaining)?,
)?;

// Mint levercoin to user
mint_tokens(
  user_levercoin_ta,
  levercoin_mint,
  levercoin_auth,
  token_program,
  levercoin_to_mint.bits,
```

```
let levercoin_to_redeem = UFix64::<N6>::new(amount);
let collateral_vault_amount =
  normalize_mint_exp(collateral_mint, collateral_vault.amount)?;

let exchange_context = ExoExchangeContext::load(
  Clock::get()?,
  collateral_vault_amount,
  exo_pair.stablecoin_mint_threshold()?,
  exo_pair.oracle_config()?,
  exo_pair.levercoin_fees,
  collateral_usd_pyth_feed,
  exo_pair.virtual_stablecoin,
  Some(levercoin_mint),
  exo_pair.rebalance_deviation_tolerance()?,
  exo_pair.sell_curve_config,
  exo_pair.buy_curve_config,
  exo_pair.levercoin_market_cap_limit()?,
)?;

let levercoin_nav = exchange_context.levercoin_redeem_nav()?;
let amount_collateral_out = exchange_context
  .exo_conversion()
  .token_to_exo(levercoin_to_redeem, levercoin_nav)?;
```

```
// Return collateral to user
redeem_collateral(
  user_collateral_ta,
  collateral_vault,
  collateral_mint,
  &vault_auth.to_account_info(),
  signer_seeds!(
    EXO_VAULT_AUTH,
    collateral_mint.key(),
    exo_pair.vault_auth_bump
  ),
  denormalize_mint_exp(collateral_mint, amount_remaining)?,
  token_program,
)?;

// Burn levercoin
burn_tokens(
  levercoin_mint,
  &user.to_account_info(),
  user_levercoin_ta,
  levercoin_to_redeem.bits,
  token_program,
)?;
```

Mitigation Suggestion

Seed new Exo levercoin markets with minimum liquidity that is permanently burned or sent to a dead address, so the first minter cannot control the initial share price.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/0599e8fba1d2d8627a39652a713667333d1eecf8.

ACC-M5 Rebalance Loss Underaccounting Can Leave the Protocol Undercollateralized When the Earn Pool Is Short

MEDIUM FIXED

Description

`settle_rebalance_pnl_lst` and `settle_rebalance_pnl_exo` pass the full rebalance loss into `absorb_loss`.

However, `absorb_loss` only burns `min(pool_balance, requested_loss)` and returns the amount actually burned. The settle handlers then burn virtual stablecoin only by that returned amount.

If `loss > pool_balance`, the unabsorbed gap is never accounted for. The rebalance swap already moved collateral based on the full loss, but virtual stablecoin is only reduced by the smaller burned amount.

This can silently leave the system undercollateralized, especially if repeated rebalance swaps occur while the earn pool has low HYUSD liquidity.

Location

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_rebalance_pnl_lst.rs#L85-L91

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_rebalance_pnl_exo.rs#L95-L101

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/absorb_loss.rs#L72-L93

Relevant Code

```
RebalancePnl::Loss(loss) => {
  // Absorb loss into earn pool
  let event = absorb_loss((&*self).into(), self.earn_pool, loss.bits)?;

  // Burn virtual stablecoin
  let stablecoin_burned = event.amount_stablecoin_burned.try_into()?;
  self.hylo.virtual_stablecoin.burn(stablecoin_burned)?;
```

```
RebalancePnl::Loss(loss) => {
  // Absorb loss into earn pool
  let event = absorb_loss((&*self).into(), self.earn_pool, loss.bits)?;

  // Burn virtual stablecoin
  let stablecoin_burned = event.amount_stablecoin_burned.try_into()?;
  self.exo_pair.virtual_stablecoin.burn(stablecoin_burned)?;
```

```
let pool_balance = UFix64::<N6>::new(stablecoin_pool.amount);
let requested_loss = UFix64::<N6>::new(amount);

// Burn either entire pool or requested loss
let amount_stablecoin_burned = pool_balance.min(requested_loss);
burn_tokens_signed(
  stablecoin_mint,
  pool_auth,
  stablecoin_pool,
  amount_stablecoin_burned.bits,
  token_program,
  signer_seeds!(POOL_AUTH, pool_config.pool_auth_bump),
)?;
```

Mitigation Suggestion

Do not silently settle a partial loss. Require `amount_stablecoin_burned == loss` and revert when the earn pool cannot absorb the full loss.

```
let stablecoin_burned: UFix64<N6> =
  event.amount_stablecoin_burned.try_into()?;

require_eq!(
  stablecoin_burned.bits,
  loss.bits,
  ErrorCode::InsufficientEarnPoolLiquidity
);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/dbd37ecc3e6ca72d6ca914f3dc3a4c941b2cf506.

ACC-M6 Stablecoin Profit Mint Can Push Rebalance Settlement Into Depeg After the Check

MEDIUM FIXED

Description

`SettleRebalancePnlExo` and `SettleRebalancePnlLst` check that the current rebalance mode is not `Depeg` before applying PnL.

However, on the profit path, both handlers mint virtual stablecoin and real stablecoin to the earn pool after this check. This increases stablecoin liabilities and can move the pair into `Depeg` by the end of the instruction.

As a result, settlement can pass the initial Depeg check but still leave the protocol in Depeg after the stablecoin mint.

Location

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_rebalance_pnl_exo.rs#L63-L86

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_rebalance_pnl_lst.rs#L54-L77

Relevant Code

```
// Disabled during Depeg
require!(
  exchange_context.rebalance_mode() != RebalanceMode::Depeg,
  ErrorCode::SettleRebalancePnlDisabled
);

match pnl {
  RebalancePnl::Profit(profit) => {
    // Mint profit in virtual stablecoin
    self.exo_pair.virtual_stablecoin.mint(profit)?;

    // Mint profit stablecoin to earn pool
    mint_tokens(
      self.stablecoin_pool,
      self.stablecoin_mint,
      self.stablecoin_mint_auth,
      self.token_program,
      profit.bits,
      signer_seeds!(
        MINT_AUTH,
        self.stablecoin_mint.key(),
        self.hylo.stablecoin_auth_bump
      ),
    )?;
```

```
// Disabled during Depeg
require!(
  exchange_context.rebalance_mode() != RebalanceMode::Depeg,
  ErrorCode::SettleRebalancePnlDisabled
);

match pnl {
  RebalancePnl::Profit(profit) => {
    // Mint profit in virtual stablecoin
    self.hylo.virtual_stablecoin.mint(profit)?;

    // Mint profit stablecoin to earn pool
    mint_tokens(
      self.stablecoin_pool,
```

```
      self.stablecoin_mint,
      self.stablecoin_mint_auth,
      self.token_program,
      profit.bits,
      signer_seeds!(
        MINT_AUTH,
        self.stablecoin_mint.key(),
        self.hylo.stablecoin_auth_bump
      ),
    )?;
```

Mitigation Suggestion

Check the projected rebalance mode after applying the PnL, or recompute the exchange context after minting virtual stablecoin and require that the final mode is not `Depeg`.

The profit mint should be rejected or capped if it would leave the pair in `Depeg`.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/d325a49585617a4d88bd17b65f83c1e61196142c.

ACC-M7 Stale Collateral Vault Balance Can Bypass Depeg Check During Exo PnL Settlement

MEDIUM FIXED

Description

`settle_rebalance_pnl_exo` reads `collateral_vault.amount` from an Anchor `Account<TokenAccount>`. This value is cached when the instruction starts and is not automatically updated after a CPI mutates the token account.

In `handle_usdc_to_exo`, collateral is withdrawn from `collateral_vault` before `SettleRebalancePnlExo::handle` is called. Because the vault is not reloaded, settlement uses the pre-swap collateral amount to build `ExoExchangeContext`. If the withdrawal pushes the pair into Depeg, the stale context can still pass the Depeg gate and commit rebalance PnL.

The same stale-context issue exists in `handle_exo_to_usdc` after depositing collateral before settlement. That direction increases CR, so it is not the Depeg-bypass case, but the settlement context is still stale.

Location

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_rebalance_pnl_exo.rs#L45-L67

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_exo_usdc.rs#L279-L311

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_exo_usdc.rs#L434-L476

Relevant Code

```
let collateral_amount =
  normalize_mint_exp(self.collateral_mint, self.collateral_vault.amount)?;

let exchange_context = ExoExchangeContext::load(
  Clock::get()?,
  collateral_amount,
  self.exo_pair.stablecoin_mint_threshold()?,
  self.exo_pair.oracle_config()?,
  self.exo_pair.levercoin_fees,
  self.collateral_usd_pyth_feed,
  self.exo_pair.virtual_stablecoin,
  None,
  self.exo_pair.rebalance_deviation_tolerance()?,
  self.exo_pair.sell_curve_config,
  self.exo_pair.buy_curve_config,
  self.exo_pair.levercoin_market_cap_limit()?,
)?;

// Disabled during Depeg
require!(
  exchange_context.rebalance_mode() != RebalanceMode::Depeg,
  ErrorCode::SettleRebalancePnlDisabled
);
```

```
// Withdraw collateral to user
redeem_collateral(
  user_collateral_ta,
  collateral_vault,
  collateral_mint,
  &vault_auth.to_account_info(),
  signer_seeds!(
    EXO_VAULT_AUTH,
```

```
    collateral_mint.key(),
    exo_pair.vault_auth_bump
  ),
  denormalize_mint_exp(collateral_mint, collateral_out)?,
  token_program,
)?;

// Settle `PnL`
SettleRebalancePnlExo {
  collateral_vault,
  // ...
}
.handle(rebalance_pnl)?;
```

```
// Deposit collateral to vault
deposit_collateral(
  user_collateral_ta,
  collateral_vault,
  collateral_mint,
  user,
  token_program,
  denormalize_mint_exp(collateral_mint, amount_collateral)?,
)?;

// Settle `PnL`
SettleRebalancePnlExo {
  collateral_vault,
  // ...
}
.handle(rebalance_pnl)?;
```

Mitigation Suggestion

Reload `collateral_vault` after any CPI that changes its balance and before building the settlement exchange context.

```
redeem_collateral(...)?;
collateral_vault.reload()?;

SettleRebalancePnlExo {
  collateral_vault,
  // ...
}
.handle(rebalance_pnl)?;
```

```
deposit_collateral(...)?;
collateral_vault.reload()?;

SettleRebalancePnlExo {
  collateral_vault,
  // ...
}
.handle(rebalance_pnl)?;
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/2e33706597fd5aa91e6bf4004132bcbe63b0a9a7.

ACC-M8 Marinade-Family LSTs Can Be Mispriced in Rebalance Swaps

MEDIUM FIXED

Description

The LST registry stores `lst_header.price_sol` using the Sanctum calculator path. That path supports all registered LST families because it selects the calculator from `lst_header.stake_program`.

However, the LST/USDC rebalance swap path does not use that family-aware price path. It always parses `pool_state` as an SPL stake pool through `SplStakePool::from_bytes`, which reads fixed byte offsets for `total_lamports` and `pool_token_supply`.

This works for SPL-family pools, but not for Marinade-family pools. For a Marinade pool, those offsets point to unrelated bytes inside the account. The read can still succeed because the account is large enough, but the decoded values are not the real lamports or supply. As a result, `true_price()` can return an incorrect price and rebalance swaps can be mispriced.

This issue is latent cause only JitoSOL and HyloSOL are registered, but it becomes active if a Marinade-family LST is added.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L34-L44

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L134-L147

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L286-L295

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L239-L245

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L424-L430

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/lst/stake_pool.rs#L11-L15

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/lst/stake_pool.rs#L31-L69

Relevant Code

```
/// lst_registry.rs L34-L44
macro_rules! sanctum_context {
  ($registry:expr, $lst_header:expr) => {
    match $lst_header.stake_program {
      Spl => &$registry.spl,
      SanctumSpl => &$registry.sanctum_spl,
      SanctumSplMulti => &$registry.sanctum_spl_multi,
      Marinade => &$registry.marinade,
    }
  };
}
```

```
/// lst_registry.rs L286-L295
self.mint = mint.key();
self.vault = vault.key();
self.pool_state = pool_state.key();
self.stake_program = sanctum_context.try_into()?;
```

```
self.price_sol = LstSolPrice::new(
  sanctum_context.lst_sol_price(mint, pool_state)?.into(),
  current_epoch()?,
);
self.prev_price_sol = self.price_sol;
self.last_yield_harvest_epoch = config.yield_harvest_cache.epoch;
```

```
/// swap_lst_usdc.rs L239-L245
// True LST price from stake pool
let stake_pool = SplStakePool::from_bytes(&pool_state.data.borrow())?;
let true_price = stake_pool.true_price()?;

// Discount true price by rebalancing fee
let adjusted_lst_sol_price =
  true_price.adjust_price(lst_header.rebalance_fee()?)?;
```

```
/// stake_pool.rs L11-L15
const TOTAL_LAMPORTS_OFFSET: usize = 258;
const POOL_TOKEN_SUPPLY_OFFSET: usize = TOTAL_LAMPORTS_OFFSET + U64_SIZE;
const LAST_UPDATE_EPOCH_OFFSET: usize = POOL_TOKEN_SUPPLY_OFFSET + U64_SIZE;
const U64_SIZE: usize = size_of::<u64>();
```

```
/// stake_pool.rs L31-L69
pub fn from_bytes(data: &[u8]) -> Result<SplStakePool> {
  let total_lamports = data
    .get(TOTAL_LAMPORTS_OFFSET..TOTAL_LAMPORTS_OFFSET + U64_SIZE)
    .and_then(|b| b.try_into().ok())
    .map(u64::from_le_bytes)
    .map(UFix64::new)
    .ok_or(ProgramError::InvalidAccountData)?;

  let pool_token_supply = data
    .get(POOL_TOKEN_SUPPLY_OFFSET..POOL_TOKEN_SUPPLY_OFFSET + U64_SIZE)
    .and_then(|b| b.try_into().ok())
    .map(u64::from_le_bytes)
    .map(UFix64::new)
    .ok_or(ProgramError::InvalidAccountData)?;

  let price = self
    .total_lamports
    .mul_div_floor(UFix64::one(), self.pool_token_supply)
    .ok_or(CoreError::StakePoolDivByZero)?;
```

Mitigation Suggestion

Do not parse every LST pool as an SPL stake pool. Use the family-aware Sanctum calculator path, or branch on `lst_header.stake_program` and use the correct parser for each supported family.

Also validate the pool account owner/layout before reading fixed offsets.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/564473b1c7a5a23c2f26b026b681527c6b9da484.

ACC-M9 Missed Borrow-Rate Harvests Are Not Accrued Across Elapsed Epochs

MEDIUM FIXED

Description

`harvest_borrow_rate` only applies one epoch of borrow rate, even if multiple epochs have passed since the last harvest.

The instruction checks that `borrow_rate_harvest_cache` is stale, but it does not calculate `current_epoch - cache.epoch`. After harvesting, it updates the cache directly to the current epoch. If the Exo pair is paused for many epochs, or if no one cranks the harvest instruction, the protocol collects only one epoch of interest and permanently skips the missed epochs.

This undercharges borrow-rate revenue and can leave the protocol with less HYUSD minted to the earn pool than intended.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L161-L167

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L211-L233

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L245-L250

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/borrow_rate.rs#L57-L62

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/yields.rs#L104-L120

Relevant Code

```
/// harvest_borrow_rate.rs L161-L167
require!(
  exo_pair
    .borrow_rate_harvest_cache
    .is_stale(exchange_context.clock.epoch),
  ErrorCode::BorrowRateHarvestAlreadyRun
);
```

```
/// harvest_borrow_rate.rs L211-L233
let stablecoin_to_harvest = {
  let raw_stablecoin = exo_pair
    .borrow_rate_config
    .apply_borrow_rate(levercoin_market_cap)?
    .checked_convert()
    .ok_or(ErrorCode::TokenAmountPrecisionError)?;

  exchange_context
    .validate_stablecoin_amount(raw_stablecoin)
    .or_else(|_| exchange_context.max_mintable_stablecoin())?
};

exo_pair.virtual_stablecoin.mint(stablecoin_to_harvest)?;
```

```
/// harvest_borrow_rate.rs L245-L250
exo_pair.borrow_rate_harvest_cache.update(
  earn_pool_cap,
```

```
  amount_remaining,
  exchange_context.clock.epoch,
)?;
```

```
/// borrow_rate.rs L57-L62
pub fn apply_borrow_rate(&self, amount: UFix64<N9>) -> Result<UFix64<N9>> {
  let rate = self.rate()?;
  amount
    .mul_div_floor(rate, UFix64::one())
    .ok_or(BorrowRateApply.into())
}
```

```
/// yields.rs L104-L120
pub fn update(
  &mut self,
  stability_pool_cap: UFix64<N6>,
  stablecoin_to_pool: UFix64<N6>,
  epoch: u64,
) -> Result<()> {
  self.epoch = epoch;
  self.stability_pool_cap = stability_pool_cap.into();
  self.stablecoin_to_pool = stablecoin_to_pool.into();
  Ok(())
}

pub fn is_stale(&self, current_epoch: u64) -> bool {
  self.epoch < current_epoch
}
```

Mitigation Suggestion

Calculate the elapsed epochs since the last borrow-rate harvest and apply the borrow rate over that full interval before updating the cache.

```
let elapsed_epochs = exchange_context
  .clock
  .epoch
  .checked_sub(exo_pair.borrow_rate_harvest_cache.epoch)
  .ok_or(ErrorCode::MathOverflow)?;

let stablecoin_to_harvest = exo_pair
  .borrow_rate_config
  .apply_borrow_rate_for_epochs(levercoin_market_cap, elapsed_epochs)?;
```

The implementation should decide whether the rate should accrue linearly or compound across missed epochs.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/1a0ea31312e532f4e63362c431d6e9bd062af2fa.

ACC-M10 Address Update Process Can Accept Malicious Admin Address Using Bait And Switch

MEDIUM FIXED

Description

We found that the implemented address update system is not sound and a malicious admin could sneak a malicious address update past the upgrade authority.

The current implementation works like this:

1. Current admin calls `ProposeAddressUpdate`, creating a new `AddressUpdateProposal` at a PDA derived only from the `AddressField` enum. The proposal contains the new address and a TTL. 2. The proposal can be canceled by the admin at any time, closing the account. 3. The proposal has to be approved by the upgrade authority by signing an instruction calling `ApproveAddressUpdate` before the proposal expires. 4. Once the proposal is approved, the new address has to sign `AcceptAddressUpdate` to finish the update.

This process is vulnerable to a bait-and-switch attack.

A is the old Admin. B is a new, verified good address for a new admin. C is a malicious address that the upgrade authority doesn't approve of.

1. Admin proposes: Change admin from A to B 2. Upgrade authority multisig creates a proposal to approve the new admin and starts voting on it 3. Just before voting ends, attacker cancels the proposal, and replaces it with a proposal to upgrade admin from A to C. 4. Multisig proposal is executed, approving the same proposal but for the wrong address 5. C accepts the new admin role

This is because the approval instruction does not bind the approval to a specific new address. It binds the approval only to the reusable upgrade PDA.

Location

https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/approve_address_update.rs#L11-L36

Relevant Code

```
/// approve_address_update.rs L11-L36
#[instruction(address_field: AddressField)]
pub struct ApproveAddressUpdate<'info> {
  pub upgrade_authority: Signer<'info>,

  #[account(
    mut,
    seeds = [
      ADDRESS_UPDATE_PROPOSAL,
      address_field.as_ref(),
    ],
    bump,
  )]
  pub proposal: Account<'info, AddressUpdateProposal>,

  #[account(
    constraint =
      program_data.upgrade_authority_address == Some(upgrade_authority.key())
        @ ErrorCode::AddressChangeUpgradeAuthority
  )]
  pub program_data: Account<'info, ProgramData>,

  #[account(
    constraint = hylo_exchange.programdata_address()? == Some(program_data.key())
```

```
  )]
  pub hylo_exchange: Program<'info, HyloExchange>,
}
```

Mitigation Suggestion

The easiest way to fix this is to require that the `proposal.new_address` is passed as an account to the `ApproveAddressUpdate` instruction, and validated that it matches the proposal.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/5cc2260d223d84258894306a8eadfa09bd9c0453.

ACC-M11 Exact Drawdown Repayments Can Cause Depeg Oscillation

MEDIUM FIXED

Description

The virtual stablecoin settlement logic burns or repays the exact amount needed based on the current overhang or surplus.

In Depeg mode, the protocol burns `min(overhang, pool_balance)`. In any non-Depeg mode, including SellZone1 and SellZone2, it repays `min(surplus, outstanding)`. This is too strict around the Depeg boundary.

Repaying the exact surplus can move the system back to the boundary, making it easy for a small adverse price change to enter Depeg again. This can create repeated drawdown -> repay -> drawdown transitions and destabilize flows that depend on whether we are in Depeg mode or drawdown is outstanding.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L116-L119

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L127-L150

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L168-L186

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_exo.rs#L144-L147

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_exo.rs#L155-L184

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_exo.rs#L202-L220

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/mod.rs#L226-L247

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/rebalance/mode.rs#L75-L86

Relevant Code

```
/// settle_virtual_stablecoin_lst.rs L116-L119
match exchange_context.rebalance_mode() {
  RebalanceMode::Depeg => Self::drawdown(accounts, &exchange_context),
  _ => Self::repay(accounts, &exchange_context),
}
```

```
/// settle_virtual_stablecoin_lst.rs L127-L150
let overhang = exchange_context.virtual_stablecoin_overhang()?;
let pool_balance = UFix64::<N6>::new(accounts.stablecoin_pool.amount);
let stablecoin_to_burn = overhang.min(pool_balance);

accounts.hylo.virtual_stablecoin.burn(stablecoin_burned)?;
accounts.hylo.pool_drawdown.drawdown(stablecoin_burned)?;
```

```
/// settle_virtual_stablecoin_lst.rs L172-L186
let surplus = exchange_context.virtual_stablecoin_surplus()?;
let outstanding = accounts.hylo.pool_drawdown.outstanding()?;
let amount_to_mint = surplus.min(outstanding);
```

```
accounts.hylo.virtual_stablecoin.mint(amount_to_mint)?;
accounts.hylo.pool_drawdown.repay(amount_to_mint)?;
```

```
/// settle_virtual_stablecoin_exo.rs L144-L147
match exchange_context.rebalance_mode() {
  RebalanceMode::Depeg => Self::drawdown(accounts, &exchange_context),
  _ => Self::repay(accounts, &exchange_context),
}
```

```
/// exchange_context/mod.rs L226-L247
fn virtual_stablecoin_overhang(&self) -> Result<UFix64<N6>> {
  let tvl = self.total_value_locked()?;
  let virtual_stablecoin = self.virtual_stablecoin_supply()?;
  tvl
    .checked_convert::<N6>()
    .and_then(|tvl| virtual_stablecoin.checked_sub(&tvl))
    .ok_or(VirtualStablecoinOverhang.into())
}

fn virtual_stablecoin_surplus(&self) -> Result<UFix64<N6>> {
  let tvl = self.total_value_locked()?;
  let virtual_stablecoin = self.virtual_stablecoin_supply()?;
  tvl
    .checked_convert::<N6>()
    .and_then(|tvl| tvl.checked_sub(&virtual_stablecoin))
    .ok_or(VirtualStablecoinSurplus.into())
}
```

```
/// mode.rs L75-L86
RebalanceMode::Depeg => CrRange::new(
  Bound::Included(UFix64::constant(0)),
  Bound::Excluded(UFix64::constant(1_000_000_000)),
),
RebalanceMode::SellZone2 => CrRange::new(
  Bound::Included(UFix64::constant(1_000_000_000)),
  Bound::Excluded(UFix64::constant(1_200_000_000)),
),
RebalanceMode::SellZone1 => CrRange::new(
  Bound::Included(UFix64::constant(1_200_000_000)),
  Bound::Excluded(UFix64::constant(1_350_000_000)),
),
```

Mitigation Suggestion

Do not repay drawdown in every non-Depeg mode. Gate repayment to safer modes, or cap the repayment so the projected collateral ratio remains above a safety threshold.

For example, repayment can require the projected state to stay above the SellZone1 upper bound or another configured buffer.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/40eb78aefdee63a092201ac219e8ae4fdea9a3ce .

ACC-M12 Missing Harvest Gates Can Double-Count Drawdown Repayments

MEDIUM FIXED

Description

`settle_virtual_stablecoin_lst` and `settle_virtual_stablecoin_exo` can repay drawdown without first checking that the current epoch's harvest has run.

For LSTs, this means a caller can update LST prices, repay drawdown from the resulting virtual stablecoin surplus, and then call `harvest_yield` to harvest. This can over-credit the earn pool and inflate virtual stablecoin accounting. If `harvest_yield` could pay the drawdown and repay, it should be called first.

The Exo path has the same missing ordering check for the borrow/funding-rate harvest.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L92-L119

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L181-L199

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_exo.rs#L117-L147

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_exo.rs#L202-L220

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_yield.rs#L168-L203

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L211-L250

Relevant Code

```
/// settle_virtual_stablecoin_lst.rs L116-L119
match exchange_context.rebalance_mode() {
  RebalanceMode::Depeg => Self::drawdown(accounts, &exchange_context),
  _ => Self::repay(accounts, &exchange_context),
}
```

```
/// settle_virtual_stablecoin_lst.rs L181-L199
fn repay(
  accounts: &mut SettleVirtualStablecoinLst<'info>,
  exchange_context: &LstExchangeContext<Clock>,
) -> Result<SettleVirtualStablecoinLstEvent> {
  let surplus = exchange_context.virtual_stablecoin_surplus()?;
  let outstanding = accounts.hylo.pool_drawdown.outstanding()?;
  let amount_to_mint = surplus.min(outstanding);

  accounts.hylo.virtual_stablecoin.mint(amount_to_mint)?;
  accounts.hylo.pool_drawdown.repay(amount_to_mint)?;

  mint_tokens(
    &accounts.stablecoin_pool,
```

```
/// settle_virtual_stablecoin_exo.rs L144-L147
match exchange_context.rebalance_mode() {
  RebalanceMode::Depeg => Self::drawdown(accounts, &exchange_context),
  _ => Self::repay(accounts, &exchange_context),
}
```

```
/// harvest_yield.rs L168-L203
let mut lst_registry = LstRegistry::load(remaining_accounts, lst_registry)?;

let harvestable_sol = lst_registry.total_sol_to_harvest()?;

let stablecoin_harvest = {
  let raw_stablecoin =
    exchange_context.sol_to_stablecoin(harvestable_sol)?;
  exchange_context
    .validate_stablecoin_amount(raw_stablecoin)
    .or_else(|_| exchange_context.max_mintable_stablecoin())?
};

hylo.virtual_stablecoin.mint(fees_extracted)?;
hylo.virtual_stablecoin.mint(stablecoin_to_pool)?;
```

```
/// harvest_borrow_rate.rs L211-L250
let stablecoin_to_harvest = {
  let raw_stablecoin = exo_pair
    .borrow_rate_config
    .apply_borrow_rate(levercoin_market_cap)?
    .checked_convert()
    .ok_or(ErrorCode::TokenAmountPrecisionError)?;
  exchange_context
    .validate_stablecoin_amount(raw_stablecoin)
    .or_else(|_| exchange_context.max_mintable_stablecoin())?
};

exo_pair.virtual_stablecoin.mint(stablecoin_to_harvest)?;

exo_pair.borrow_rate_harvest_cache.update(
  earn_pool_cap,
  amount_remaining,
  exchange_context.clock.epoch,
)?;
```

Mitigation Suggestion

Require the relevant harvest to be completed before virtual stablecoin settlement.

```
let clock = Clock::get()?;

require_eq!(
  accounts.hylo.yield_harvest_cache.epoch,
  clock.epoch,
  ErrorCode::YieldHarvestHasNotRun
);
```

```
let clock = Clock::get()?;

require_eq!(
  accounts.exo_pair.borrow_rate_harvest_cache.epoch,
  clock.epoch,
  ErrorCode::BorrowRateHarvestHasNotRun
);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/6dbd933a44c07b65d75b4850630f795d57c38876.

ACC-M13 LP Accounting Ignores Drawdown Debt and Misprices sHYUSD

MEDIUM ACKNOWLEDGED

Description

The earn-pool deposit and withdraw paths calculate LP value using only the current `stablecoin_pool.amount`.

In the new design, the earn pool can have HYUSD burned by the exchange during depeg settlement. That burned amount is tracked as pool drawdown and is expected to be repaid later when the pair exits depeg. Although this amount is not immediately liquid, it is still part of the earn pool's future claim.

Because deposits and withdrawals ignore outstanding drawdown, sHYUSD can be mispriced while drawdown exists. New depositors may receive too many LP tokens, and users withdrawing during drawdown may lose their share of the future repayment to remaining LP holders.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/user_deposit.rs#L110-L118

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/user_withdraw.rs#L125-L133

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L128-L151

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_virtual_stablecoin_lst.rs#L173-L188

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/earn_pool_math.rs#L16-L48

Relevant Code

```
/// user_deposit.rs L110-L118
let lp_token_nav = lp_token_nav(
  UFix64::new(stablecoin_pool.amount),
  UFix64::new(lp_token_mint.supply),
)?;

let stablecoin_deposited = UFix64::new(amount_stablecoin);
let lp_token_amount = lp_token_out(stablecoin_deposited, lp_token_nav)?;
```

```
/// user_withdraw.rs L125-L133
let lp_tokens_to_burn = UFix64::new(amount_lp_token);
let lp_token_supply = UFix64::new(lp_token_mint.supply);
let stablecoin_in_pool = UFix64::new(stablecoin_pool.amount);
let stablecoin_to_withdraw = amount_token_to_withdraw(
  lp_tokens_to_burn,
  lp_token_supply,
  stablecoin_in_pool,
)?;
```

```
/// settle_virtual_stablecoin_lst.rs L137-L151
let AbsorbLossEvent {
  amount_stablecoin_burned,
  remaining_pool_balance,
  ..
} = absorb_loss(
  (&*accounts).into(),
  &accounts.earn_pool,
```

```
  stablecoin_to_burn.bits,
)?;
let stablecoin_burned = amount_stablecoin_burned.try_into()?;

accounts.hylo.virtual_stablecoin.burn(stablecoin_burned)?;
accounts.hylo.pool_drawdown.drawdown(stablecoin_burned)?;
```

```
/// settle_virtual_stablecoin_lst.rs L173-L188
let surplus = exchange_context.virtual_stablecoin_surplus()?;
let outstanding = accounts.hylo.pool_drawdown.outstanding()?;
let amount_to_mint = surplus.min(outstanding);

accounts.hylo.virtual_stablecoin.mint(amount_to_mint)?;
accounts.hylo.pool_drawdown.repay(amount_to_mint)?;

mint_tokens(
  &accounts.stablecoin_pool,
```

Mitigation Suggestion

Price sHYUSD using both the liquid pool balance and the outstanding drawdown owed back to the earn pool.

```
let total_pool_assets = stablecoin_pool_amount
  .checked_add(total_pool_drawdown_outstanding)
  .ok_or(ErrorCode::MathOverflow)?;

let lp_token_nav = lp_token_nav(
  total_pool_assets,
  UFix64::new(lp_token_mint.supply),
)?;
```

Because drawdown is not immediately liquid, if all users withdraw during the positive drawdown state, there will be no tokens for the last withdrawals, so withdrawals need a clear design decision: either pause withdrawals while drawdown is outstanding, queue the unpaid portion, or allow partial withdrawals while preserving each user's claim on future repayments.

Remediation

The issue has been marked as acknowledged.

Hylo: We discussed this one and we do not feel it is correct to factor in a debt which may or may not be paid back into the NAV. We can surface any outstanding depeg drawdown in our risk dashboard or earn pool deposit UI so users are aware of the risks. We do also want to reward a user willing to underwrite the pool with the potential upside of the bad debt being repaid, not unlike a lending market.

ACC-M14 Mint Fee Uses Pre-Fee Amount and Overcharges Users

MEDIUM ACKNOWLEDGED

Description

We found that projected stablecoin supply for collateral ratio (CR) is computed using the pre-fee amount_lst, but the actual minted stablecoin comes from the post-fee remaining amount. This makes projected cr low, which then increases the fee rate on the curve.

So the circular dependency is resolved in a way that always penalizes the user: the system consistently charges a higher fee than it should.

Location

https://github.com/hylo-so/sdk/blob/a64243b6f2704946eb0367387880b9aa44fdfe13/hylo-core/src/exchange_context/lst.rs#L149-L177

Relevant Code

```
/// Stablecoin mint fee via interpolated curve at projected CR.
  ///
  /// # Errors
  /// * Projection overflow, interpolation, or fee extraction
  pub fn stablecoin_mint_fee(
    &self,
    lst_sol_price: &LstSolPrice,
    amount_lst: UFix64<N9>,
  ) -> Result<FeeExtract<N9>> {
    let new_sol =
      lst_sol_price.convert_lst_to_sol(amount_lst, self.clock.epoch())?;
    let new_total_sol = self
      .total_sol
      .checked_add(&new_sol)
      .ok_or(DestinationFeeSol)?;
    let new_total_stablecoin = self
      .token_conversion(lst_sol_price)?
      .lst_to_token(amount_lst, self.stablecoin_nav()?)?
      .checked_add(&self.virtual_stablecoin_supply()?)
      .ok_or(DestinationFeeStablecoin)?;
    let projected_cr = collateral_ratio(
      new_total_sol,
      self.sol_usd_price.lower,
      new_total_stablecoin,
    )?;
    self
      .stablecoin_mint_fees
      .apply_fee(projected_cr, amount_lst)
  }
```

Mitigation Suggestion

-

Remediation

Acknowledged.

ACC-M15 Mint Fee Reverts in Valid Mode1 Range (CR 130–150)

MEDIUM FIXED

Description

We found that in the new fee mechanism, the mint/redeem stablecoin fee is calculated based on the curve. In the mint path, the fee calculation reverts if the collateral ratio (CR) is less than 150.

This is problematic because minting stablecoin is allowed in Mode1, and Mode1 includes the CR range 130 (stability two)–150. As a result, minting stablecoin from LST and EXO will fail within a valid and intended operating range.

Location

https://github.com/hylo-so/sdk/blob/a64243b6f2704946eb0367387880b9aa44fdfe13/hylo-core/src/interpolated_fees.rs#L77-L78

Relevant Code

```
if cr < interp.x_min() {
      Err(CoreError::NoValidStablecoinMintFee.into())
```

Mitigation Suggestion

Allow mint fee calculation to work for the full Mode1 CR range (130–150) and prevent reverts within valid modes.

Remediation

Fixed in commit https://github.com/hylo-so/protocol/pull/274 .

ACC-M16 Using init for ATA Creation Enables Pre-Init DoS

MEDIUM FIXED

Description

We found that some instructions use Anchor init to create an ATA. This allows an attacker to create the same ATA before the instruction runs, causing the instruction to fail and creating a denial-of-service (DoS) risk.

Example in initialize_usdc.rs: usdc_fee_vault can be created by anyone ahead of time, so the real initialization will revert.

Location

https://github.com/hylo-so/protocol/blob/90b497803bc691584b77e4f380f485cd954fbaff/programs/hylo-exchange/src/instructions/initialize_usdc.rs#L49-L63

Relevant Code

```
#[account(
    init,
    payer = admin,
    associated_token::mint = usdc_mint,
    associated_token::authority = usdc_vault_auth,
  )]
  pub usdc_collateral_vault: Account<'info, TokenAccount>,


  #[account(
    init,
    payer = admin,
    associated_token::mint = usdc_mint,
    associated_token::authority = usdc_fee_auth,
  )]
  pub usdc_fee_vault: Account<'info, TokenAccount>,
```

Mitigation Suggestion

Use init_if_needed for ATA creation so the instruction succeeds even if the ATA already exists.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/85c27ee0ee01e0bca43bbd3b9fad0c5d71cd3338.

ACC-L1 Oracle Freeze Locks out Settlement While Allowing Users To Exit Earn Pool

LOW FIXED

Description

We found that during an oracle freeze event, which means that an oracle becomes rejected by the protocol due to quoting too wide or being stale, which can happen in a crash, savvy earn pool users can abuse this to their advantage. In such a situation, the settlement instructions fail because the oracle is rejected. However, at the same time, users can exit the earn pool by withdrawing, because no oracle checks are being done. This creates a window during a potential crash where earn pool losses are expected as soon as the oracle stabilizes, but earn pool holders can still exit at pre-settlement NAV.

This is potentially the strongest version of an inherent issue with the earn pool: Users can exit the pool whenever losses are predictable. This means that users can participate in an earn pool when CRs are healthy, earning a premium which is supposed to compensate them for the insurance they provide to the protocol, and exit as soon as trouble is on the horizon (protocol entering sell zones). They can earn a risk premium while cutting off their tail risk almost entirely this way.

Location

https://github.com/hylo-so/protocol/blob/6f53f619e61b009acbe3d631a7d463b0c5544e0f/programs/hylo-earn-pool/src/instructions/user_withdraw.rs#L21-L96

Relevant Code

```
/// user_withdraw.rs L21-L96
pub struct UserWithdraw<'info> {
  #[account(mut)]
  pub user: Signer<'info>,

  #[account(
    seeds = [POOL_CONFIG],
    bump,
  )]
  pub pool_config: Box<Account<'info, PoolConfig>>,

  #[account(
    seeds = [HYLO],
    seeds::program = hylo_exchange::ID,
    bump,
  )]
  pub hylo: Box<Account<'info, Hylo>>,

  #[account(
    seeds = [HYUSD],
    seeds::program = hylo_exchange::ID,
    bump = hylo.stablecoin_mint_bump,
  )]
  pub stablecoin_mint: Box<Account<'info, Mint>>,

  #[account(
    mut,
    token::mint = stablecoin_mint,
    token::authority = user,
  )]
  pub user_stablecoin_ta: Box<Account<'info, TokenAccount>>,

  /// CHECK: PDA
  #[account(
    seeds = [FEE_AUTH, stablecoin_mint.key().as_ref()],
    seeds::program = hylo_exchange::ID,
    bump,
```

```
  )]
  pub fee_auth: UncheckedAccount<'info>,

  #[account(
    mut,
    associated_token::authority = fee_auth,
    associated_token::mint = stablecoin_mint,
  )]
  pub fee_vault: Box<Account<'info, TokenAccount>>,

  #[account(
    mut,
    token::mint = lp_token_mint,
    token::authority = user,
  )]
  pub user_lp_token_ta: Box<Account<'info, TokenAccount>>,

  /// CHECK: PDA
  #[account(
    seeds = [POOL_AUTH],
    bump = pool_config.pool_auth_bump,
  )]
  pub pool_auth: UncheckedAccount<'info>,

  #[account(
    mut,
    associated_token::authority = pool_auth,
    associated_token::mint = stablecoin_mint,
  )]
  pub stablecoin_pool: Box<Account<'info, TokenAccount>>,

  #[account(
    mut,
    seeds = [STAKED_HYUSD],
    bump = pool_config.lp_token_mint_bump,
  )]
  pub lp_token_mint: Box<Account<'info, Mint>>,

  pub token_program: Program<'info, Token>,
}
```

Mitigation Suggestion

The main issue seems to be the ungated, flat fee earn pool withdrawal. The main solution is probably a long withdrawal timeout and additional exit fees on outstanding drawdowns/underwater pools.

Remediation

The issue has been acknowledged. A future fix may be implemented using a per-epoch liquidity window allowing users to withdraw X% of a pool.

ACC-L2 USDC Sell Swaps Ignore Drawdown and Can Sell Collateral While the Pair Is Effectively Depegged

LOW FIXED

Description

The USDC sell-side swap paths do not check whether pool drawdown has been repaid. Drawdown means the reported CR can be higher than the real CR because there is still outstanding debt owed back to the earn pool.

Because the swap context ignores this outstanding drawdown, the pair can appear to be in `SellZone2` even though the real state should be Depeg. In that case, the USDC swap path can continue selling collateral with a loss instead of reverting until the drawdown is repaid.

Location

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_exo_usdc.rs#L209-L267

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L392-L479

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/rebalance/pool_drawdown.rs#L37-L65

Relevant Code

```
let exchange_context = ExoExchangeContext::load(
  Clock::get()?,
  collateral_vault_amount,
  exo_pair.stablecoin_mint_threshold()?,
  exo_pair.swap_oracle_config()?,
  exo_pair.levercoin_fees,
  collateral_usd_pyth_feed,
  exo_pair.virtual_stablecoin,
  Some(levercoin_mint),
  exo_pair.rebalance_deviation_tolerance()?,
  exo_pair.sell_curve_config,
  exo_pair.buy_curve_config,
  exo_pair.levercoin_market_cap_limit()?,
)?;

// Sell-side must be active
require!(
  exchange_context.rebalance_sell_active(),
  ErrorCode::RebalanceSellInactive
);
```

```
let exchange_context = LstExchangeContext::load(
  Clock::get()?,
  &hylo.total_sol_cache,
  hylo.stablecoin_mint_threshold()?,
  hylo.swap_oracle_config()?,
  hylo.levercoin_fees,
  sol_usd_pyth_feed,
  hylo.virtual_stablecoin,
  None,
  hylo.rebalance_deviation_tolerance()?,
  hylo.lst_sell_curve_config,
  hylo.lst_buy_curve_config,
)?;

// Sell-side must be active
require!(
  exchange_context.rebalance_sell_active(),
```

```
  ErrorCode::RebalanceSellInactive
);
```

```
pub fn outstanding(&self) -> Result<UFix64<N6>> {
  self.ledger.supply()
}

pub fn is_repaid(&self) -> bool {
  *self == PoolDrawdown::default()
}
```

Mitigation Suggestion

Revert USDC sell-side swaps while drawdown is outstanding.

```
require!(
  exo_pair.pool_drawdown.is_repaid(),
  ErrorCode::DrawdownNotRepaid
);
```

```
require!(
  hylo.pool_drawdown.is_repaid(),
  ErrorCode::DrawdownNotRepaid
);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/682734097a5a95672c89502370e3ed9e4c1f2379.

ACC-L3 Stablecoin Minting Ignores Drawdown and Can Overstate Max Mintable Amount

LOW FIXED

Description

Drawdown means the reported CR can be higher than the real CR because there is outstanding debt that still needs to be repaid. The stablecoin mint paths do not check whether drawdown is repaid before minting.

As a result, `max_mintable_stablecoin` can return a larger value than intended, and users may mint more stablecoin while drawdown is outstanding. After the drawdown is later repaid, the system can end up with more stablecoin minted than intended for the real collateral state.

Location

- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_stablecoin_exo.rs#L138-L188
- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_stablecoin_lst.rs#L132-L193
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/mod.rs#L293-L338
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_math.rs#L58-L83

Relevant Code

```
let stablecoin_to_mint = {
  let requested = exchange_context
    .exo_conversion()
    .exo_to_token(amount_remaining, stablecoin_nav)?;
  exchange_context.validate_stablecoin_amount(requested)?
};

exo_pair.virtual_stablecoin.mint(stablecoin_to_mint)?;
```

```
let stablecoin_to_mint = {
  let requested = exchange_context
    .token_conversion(&lst_header.price_sol)?
    .lst_to_token(amount_remaining, stablecoin_nav)?;
  exchange_context.validate_stablecoin_amount(requested)?
};

hylo.virtual_stablecoin.mint(stablecoin_to_mint)?;
```

```
max_mintable_stablecoin(
  target,
  self.total_collateral(),
  self.collateral_usd_price().upper,
  self.virtual_stablecoin_supply()?,
)
```

Mitigation Suggestion

Do not allow stablecoin minting while drawdown is outstanding, or include outstanding drawdown in the CR and max mintable stablecoin calculation.

```
require!(
  exo_pair.pool_drawdown.is_repaid(),
  ErrorCode::DrawdownNotRepaid
);
```

```
require!(
  hylo.pool_drawdown.is_repaid(),
  ErrorCode::DrawdownNotRepaid
);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/045c94a9c9db890d135c1221ab32667ffefa18dc.

ACC-L4 Zero Free Collateral Can Make Levercoin NAV Zero and Panic Levercoin Minting

LOW FIXED

Description

Levercoin NAV is computed as `free_collateral / levercoin_supply` when levercoin supply is nonzero.

If free collateral is exactly zero, the NAV becomes zero. This can happen in Depeg mode or during recovery after Depeg when CR is exactly `1`.

That zero NAV is then used by mint and conversion paths as a divisor. This can cause levercoin minting or stablecoin-to-levercoin conversion to panic instead of handling the zero-free-collateral state explicitly.

Location

- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_math.rs#L113-L155
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_context/mod.rs#L195-L218
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/conversion.rs#L35-L44
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/conversion.rs#L80-L89
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/conversion.rs#L125-L133
- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_levercoin_exo.rs#L174-L180
- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/convert_exo_pair.rs#L194-L198

Relevant Code

```
let free_collateral =
  collateral_value.checked_sub(&stablecoin_value.checked_convert()?)?;
let nav = free_collateral.mul_div_ceil(UFix64::one(), levercoin_supply)?;
Some(nav)
```

```
let free_collateral =
  collateral_value.checked_sub(&stablecoin_value.checked_convert()?)?;
let nav = free_collateral.mul_div_floor(UFix64::one(), levercoin_supply)?;
Some(nav)
```

```
fn levercoin_mint_nav(&self) -> Result<UFix64<N9>> {
  next_levercoin_mint_nav(
    self.total_collateral(),
    self.collateral_usd_price(),
    self.virtual_stablecoin_supply()?,
    self.stablecoin_nav()?,
    self.levercoin_supply()?,
  )
  .ok_or(LevercoinNav.into())
}
```

```
amount_lst
  .mul_div_floor(self.lst_sol_price, UFix64::one())
```

```
  .and_then(|sol| sol.mul_div_floor(self.usd_sol_price.lower, token_nav))
  .map(UFix64::convert)
  .ok_or(LstToToken.into())
```

```
amount_stable
  .mul_div_floor(self.stablecoin_nav, UFix64::one())
  .and_then(|usd| {
    usd.mul_div_floor(UFix64::one(), self.levercoin_nav.upper)
  })
  .ok_or(StableToLever.into())
```

```
amount
  .mul_div_floor(self.collateral_usd_price.lower, token_nav)
  .and_then(UFix64::checked_convert::<N6>)
  .ok_or(ExoToToken.into())
```

Mitigation Suggestion

Treat zero free collateral as a special state instead of returning a zero levercoin NAV.

For example, reject levercoin minting and stablecoin-to-levercoin conversion when free collateral is zero, or route the protocol through an explicit recovery state until free collateral is positive.

```
require_gt!(
  free_collateral.bits,
  u64::MIN,
  ErrorCode::ZeroFreeCollateral
);
```

Remediation

Fixed in https://github.com/hylo-so/sdk/commit/99bea7e88b2635fed8639a6f86ee44e9839d211c.

ACC-L5 LST-to-LST Swaps Can Leave Total SOL Cache Out of Sync

LOW FIXED

Description

`swap_lst_to_lst` mutates two LST vaults but does not update `hylo.total_sol_cache`.

The instruction deposits `amount_remaining` of LST A into the LST A vault and withdraws `lst_b_out` from the LST B vault. Other instructions that mutate LST vault balances update `total_sol_cache` with the SOL value of the deposit or withdrawal, but this path skips that step.

The cache can therefore diverge from the actual SOL backing until the next price update resets it. This affects collateral ratio, rebalance mode, levercoin redeem NAV, and mint/rebalance caps.

Location

- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_to_lst.rs#L146-L158
- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_to_lst.rs#L164-L201
- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L280-L285
- https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L459-L465

Relevant Code

```
let FeeExtract {
  fees_extracted,
  amount_remaining,
} = hylo.lst_swap_config()?.apply_fee(lst_a_in)?;

// Convert remaining LST A to LST B
let lst_b_out = lst_a_header.price_sol.convert_lst_amount(
  epoch,
  amount_remaining,
  &lst_b_header.price_sol,
)?;
```

```
// User LST A to vault
deposit_collateral(
  lst_a_user_ta,
  lst_a_vault,
  lst_a_mint,
  user,
  token_program,
  amount_remaining.bits,
)?;

// Fees to vault
deposit_collateral(
  lst_a_user_ta,
  fee_vault,
  lst_a_mint,
  user,
  token_program,
  fees_extracted.bits,
)?;

// User LST B from vault
```

```
redeem_collateral(
  lst_b_user_ta,
  lst_b_vault,
  lst_b_mint,
  lst_b_vault_auth,
  signer_seeds!(VAULT_AUTH, lst_b_mint.key(), bumps.lst_b_vault_auth),
  lst_b_out.bits,
  token_program,
)?;

Ok(SwapLstToLstEvent {
```

```
let sol_in = lst_header
  .price_sol
  .convert_lst_to_sol(amount_lst, exchange_context.clock.epoch)?;
hylo
  .total_sol_cache
  .increment(sol_in, exchange_context.clock.epoch)?;
```

```
let sol_out = lst_header
  .price_sol
  .convert_lst_to_sol(lst_out, exchange_context.clock.epoch)?;
hylo
  .total_sol_cache
  .decrement(sol_out, exchange_context.clock.epoch)?;
```

Mitigation Suggestion

Update `hylo.total_sol_cache` after the LST-to-LST vault transfers, mirroring the existing `swap_lst_usdc` pattern.

```
let sol_in = lst_a_header
  .price_sol
  .convert_lst_to_sol(amount_remaining, epoch)?;

let sol_out = lst_b_header
  .price_sol
  .convert_lst_to_sol(lst_b_out, epoch)?;

hylo.total_sol_cache.increment(sol_in, epoch)?;
hylo.total_sol_cache.decrement(sol_out, epoch)?;
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/23f2de749cfe1316e32e359adc2ff8e04943fc18.

ACC-L6 Zero Levercoin Supply Can Reset NAV and Misprice Future Levercoin Minters

LOW FIXED

Description

Levercoin NAV returns 1 when levercoin supply is zero. The redeem paths allow a user to redeem and burn all outstanding levercoin, so the supply can return to zero while virtual stablecoin is still positive.

This means the protocol can have a positive virtual stablecoin balance with zero levercoin supply. From that state, future collateral price changes are not assigned to any existing levercoin holder, and the next levercoin minter can enter at the default zero-supply NAV instead of a NAV that reflects the existing collateral state.

This can create unfair or incorrect value distribution for the next levercoin minter and the protocol accounting.

This issue can reappear any time all levercoin exits and the supply returns to zero.

Location

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/exchange_math.rs#L113-L155

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_levercoin_lst.rs#L137-L177

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/redeem_levercoin_lst.rs#L130-L211

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_levercoin_exo.rs#L137-L180

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/redeem_levercoin_exo.rs#L130-L214

Relevant Code

```
pub fn next_levercoin_mint_nav(
  total_collateral: UFix64<N9>,
  usd_collateral_price: PriceRange<N9>,
  stablecoin_supply: UFix64<N6>,
  stablecoin_nav: UFix64<N9>,
  levercoin_supply: UFix64<N6>,
) -> Option<UFix64<N9>> {
  if levercoin_supply == UFix64::zero() {
    Some(UFix64::one())
  } else {
    let collateral_value = total_collateral
      .mul_div_ceil(usd_collateral_price.upper, UFix64::one())?;
    let stablecoin_value =
      stablecoin_supply.mul_div_floor(stablecoin_nav, UFix64::one())?;
    let free_collateral =
      collateral_value.checked_sub(&stablecoin_value.checked_convert()?)?;
    let nav = free_collateral.mul_div_ceil(UFix64::one(), levercoin_supply)?;
    Some(nav)
  }
}
```

```
pub fn next_levercoin_redeem_nav(
  total_collateral: UFix64<N9>,
  usd_collateral_price: PriceRange<N9>,
  stablecoin_supply: UFix64<N6>,
  stablecoin_nav: UFix64<N9>,
  levercoin_supply: UFix64<N6>,
) -> Option<UFix64<N9>> {
```

```
  if levercoin_supply == UFix64::zero() {
    Some(UFix64::one())
  } else {
    let collateral_value = total_collateral
      .mul_div_floor(usd_collateral_price.lower, UFix64::one())?;
    let stablecoin_value =
      stablecoin_supply.mul_div_ceil(stablecoin_nav, UFix64::one())?;
    let free_collateral =
      collateral_value.checked_sub(&stablecoin_value.checked_convert()?)?;
    let nav = free_collateral.mul_div_floor(UFix64::one(), levercoin_supply)?;
    Some(nav)
  }
}
```

```
let levercoin_nav = exchange_context.levercoin_mint_nav()?;
let levercoin_to_mint = exchange_context
  .token_conversion(&lst_header.price_sol)?
  .lst_to_token(amount_remaining, levercoin_nav)?;
```

```
let levercoin_nav = exchange_context.levercoin_redeem_nav()?;
let amount_lst_out = exchange_context
  .token_conversion(&lst_header.price_sol)?
  .token_to_lst(levercoin_to_redeem, levercoin_nav)?;

// ...

burn_tokens(
  levercoin_mint,
  &user.to_account_info(),
  user_levercoin_ta,
  levercoin_to_redeem.bits,
  token_program,
)?;
```

```
let levercoin_nav = exchange_context.levercoin_mint_nav()?;
let levercoin_to_mint = {
  let proposed_mint = exchange_context
    .exo_conversion()
    .exo_to_token(amount_remaining, levercoin_nav)?;
```

```
let levercoin_nav = exchange_context.levercoin_redeem_nav()?;
let amount_collateral_out = exchange_context
  .exo_conversion()
  .token_to_exo(levercoin_to_redeem, levercoin_nav)?;

// ...

burn_tokens(
  levercoin_mint,
  &user.to_account_info(),
  user_levercoin_ta,
  levercoin_to_redeem.bits,
  token_program,
)?;
```

Mitigation Suggestion

Seed each levercoin market with minimum liquidity that is permanently burned or sent to a dead address. This prevents levercoin supply from returning to zero and avoids resetting NAV to the default value.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/0599e8fba1d2d8627a39652a713667333d1eecf8.

ACC-L7 Hardcoded 1-Second Oracle Window Can Break Rebalance Swaps

LOW FIXED

Description

Rebalance swap paths use `swap_oracle_config()`, which hardcodes `interval_secs = 1`.

Normal operations and the self-CPI PnL settlement paths use the configurable `oracle_interval_secs`. This makes rebalance swaps much stricter than the rest of the protocol.

The Pyth freshness check requires `publish_time + interval_secs >= clock_time`, so a one-second window can fail during normal oracle propagation or update delays with `PythOracleOutdated`. If the slot interval check is also corrected, the slot freshness check will become similarly strict. This creates unnecessary liveness risk on the exact paths used for rebalancing.

Location

- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/hylo.rs#L351-L365
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/exo_pair.rs#L145-L158
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L207-L213
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_lst_usdc.rs#L394-L400
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_exo_usdc.rs#L209-L215
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_exo_usdc.rs#L367-L373
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/settle_rebalance_pnl_lst.rs#L47-L52
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/pyth.rs#L137-L153

Relevant Code

```
/// hylo.rs L351-L365
pub fn oracle_config(&self) -> Result<OracleConfig> {
  Ok(OracleConfig::new(
    self.oracle_interval_secs,
    self.oracle_conf_tolerance.try_into()?,
  ))
}

pub fn swap_oracle_config(&self) -> Result<OracleConfig> {
  Ok(OracleConfig::new(
    1u64,
    self.oracle_conf_tolerance.try_into()?,
  ))
}
```

```
/// exo_pair.rs L145-L158
pub fn oracle_config(&self) -> Result<OracleConfig> {
  Ok(OracleConfig::new(
    self.oracle_interval_secs,
    self.oracle_conf_tolerance.try_into()?,
  ))
```

```
}

pub fn swap_oracle_config(&self) -> Result<OracleConfig> {
  Ok(OracleConfig::new(
    1u64,
    self.oracle_conf_tolerance.try_into()?,
  ))
}
```

```
/// swap_lst_usdc.rs L207-L213
let exchange_context = LstExchangeContext::load(
  Clock::get()?,
  &hylo.total_sol_cache,
  hylo.stablecoin_mint_threshold()?,
  hylo.swap_oracle_config()?,
  hylo.levercoin_fees,
  sol_usd_pyth_feed,
```

```
/// swap_exo_usdc.rs L209-L215
let exchange_context = ExoExchangeContext::load(
  Clock::get()?,
  collateral_vault_amount,
  exo_pair.stablecoin_mint_threshold()?,
  exo_pair.swap_oracle_config()?,
  exo_pair.levercoin_fees,
  collateral_usd_pyth_feed,
```

```
/// settle_rebalance_pnl_lst.rs L47-L52
let exchange_context = LstExchangeContext::load(
  Clock::get()?,
  &self.hylo.total_sol_cache,
  self.hylo.stablecoin_mint_threshold()?,
  self.hylo.oracle_config()?,
```

```
/// pyth.rs L137-L153
fn validate_publish_time(
  publish_time: i64,
  oracle_interval: u64,
  clock_time: i64,
) -> Result<()> {
  let (publish_time, clock_time) =
    if publish_time.is_positive() && clock_time.is_positive() {
      Ok((publish_time.unsigned_abs(), clock_time.unsigned_abs()))
    } else {
      Err(PythOracleNegativeTime)
    }?;
  if publish_time.saturating_add(oracle_interval) >= clock_time {
    Ok(())
  } else {
    Err(PythOracleOutdated.into())
  }
```

Mitigation Suggestion

Use the configurable oracle interval for rebalance swaps, or add a separate configurable rebalance oracle interval instead of hardcoding one second.

```
pub fn swap_oracle_config(&self) -> Result<OracleConfig> {
  self.oracle_config()
}
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/fb403f38fe46e1620d4879d2b12672247dbc5489

ACC-L8 No-op LST Harvest Leaves Per-LST Harvest Cursors Stale

LOW FIXED

Description

In Sell and Depeg modes, `harvest_yield` calls `commit_noop_harvest`. This updates the global `hylo.yield_harvest_cache` for the current epoch, but it does not advance each `LstHeader.last_yield_harvest_epoch`.

Normal harvest updates both the global cache and the per-LST harvest cursors through `LstRegistry::total_sol_to_harvest`.

This creates inconsistent state: the protocol marks yield harvest as completed globally, so LST operations can continue, while the individual LST headers still show that yield has not been harvested for that epoch.

Location

- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_yield.rs#L137-L152
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_yield.rs#L252-L269
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L174-L185
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/mint_stablecoin_lst.rs#L153-L158

Relevant Code

```
/// harvest_yield.rs L137-L152
match exchange_context.rebalance_mode() {
  Neutral | BuyZone1 | BuyZone2 => Self::commit_normal_harvest(
    hylo,
    stablecoin_mint,
    stablecoin_auth,
    stablecoin_fee_vault,
    stablecoin_pool,
    lst_registry,
    token_program,
    remaining_accounts,
    &exchange_context,
    earn_pool_cap,
  ),
  SellZone1 | SellZone2 | Depeg => {
    Self::commit_noop_harvest(hylo, &exchange_context, earn_pool_cap)
  }
}
```

```
/// harvest_yield.rs L252-L269
fn commit_noop_harvest(
  hylo: &mut Box<Account<'info, Hylo>>,
  exchange_context: &LstExchangeContext<Clock>,
  earn_pool_cap: UFix64<N6>,
) -> Result<HarvestYieldEvent> {
  hylo.yield_harvest_cache.update(
    earn_pool_cap,
    UFix64::zero(),
    exchange_context.clock.epoch,
  )?;
```

```
/// lst_registry.rs L174-L185
pub fn total_sol_to_harvest(&mut self) -> Result<UFix64<N9>> {
  self
    .registry
    .iter_mut()
    .try_fold(UFix64::zero(), |acc, lst| {
      let current_epoch = current_epoch()?;
      if lst.lst_header.last_yield_harvest_epoch < current_epoch {
        lst.lst_header.last_yield_harvest_epoch = current_epoch;
        lst.lst_header.exit(&HyloExchange::id())?;
        let appreciation = lst.lst_header.sol_price_appreciation()?;
```

```
/// mint_stablecoin_lst.rs L153-L158
require_eq!(
  hylo.yield_harvest_cache.epoch,
  exchange_context.clock.epoch,
  ErrorCode::YieldHarvestHasNotRun
);
```

Mitigation Suggestion

Keep global and per-LST harvest state consistent. If no-op harvest means the epoch should be considered harvested, then `commit_noop_harvest` should also advance each `LstHeader.last_yield_harvest_epoch`.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/8c095a23821ccc4f62bcae4224353717f573ec1e.

ACC-L9 Dust xSOL Can Permanently Block Levercoin Pool Deprecation

LOW FIXED

Description

`deprecate_levercoin_pool` requires `levercoin_pool.amount` to be zero before the token account can be closed.

A malicious user can send a small amount of xSOL, even 1 token, directly to the pool token account. This makes `levercoin_pool.amount != 0`, causing the instruction to revert and blocking the deprecation process.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/deprecate_levercoin_pool.rs#L42-L48

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/deprecate_levercoin_pool.rs#L55-L65

Relevant Code

```
/// deprecate_levercoin_pool.rs L42-L48
#[account(
  mut,
  close = admin,
  associated_token::mint = levercoin_mint,
  associated_token::authority = pool_auth,
)]
pub levercoin_pool: Account<'info, TokenAccount>,
```

```
/// deprecate_levercoin_pool.rs L55-L65
impl DeprecateLevercoinPool<'_> {
  pub fn handle(
    DeprecateLevercoinPool { levercoin_pool, .. }: &mut DeprecateLevercoinPool,
  ) -> Result<()> {
    require_eq!(
      levercoin_pool.amount,
      u64::MIN,
      ErrorCode::LevercoinPoolNonzero
    );
    Ok(())
  }
}
```

Mitigation Suggestion

Drain any remaining xSOL from `levercoin_pool` to an admin-controlled token account before closing the account.

```
redeem_collateral(
  admin_levercoin_ta,
  levercoin_pool,
  levercoin_mint,
  pool_auth,
  signer_seeds!(POOL_AUTH, pool_config.pool_auth_bump),
  levercoin_pool.amount,
  token_program,
)?;
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/637045c7db54ce071d92c9c189ad0ae88e189f15.

ACC-L10 Admin Transfer Can Brick Payer-Based Admin Instructions

LOW FIXED

Description

The address update flow allows `AddressField::Admin` to be set to any `new_address` without checking whether the new admin can act as a system payer.

This is risky because several admin instructions use `admin` as the payer for new accounts. If the admin is changed to a non-system-owned account, some multisig pda accounts, future instructions that need `payer = admin` can fail, causing a denial of service for admin operations.

Note that, unlike the v1 upgrade-authority flow, the v2 admin initiates address proposals and pays for the proposal PDA.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/propose_address_update.rs#L21-L31

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/address_update_proposal.rs#L49-L68

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/hylo.rs#L143-L154

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/register_exo.rs#L43-L50

Relevant Code

```
/// propose_address_update.rs L21-L31
#[account(
  init,
  space = AddressUpdateProposal::INIT_SPACE + 8,
  payer = admin,
  seeds = [
    ADDRESS_UPDATE_PROPOSAL,
    address_field.as_ref(),
  ],
  bump,
)]
pub proposal: Account<'info, AddressUpdateProposal>,
```

```
/// address_update_proposal.rs L49-L68
pub fn init(
  &mut self,
  hylo: &Account<'_, Hylo>,
  clock: &Clock,
  address_field: AddressField,
  new_address: Pubkey,
  ttl_secs: u64,
) -> Result<()> {
  require_neq!(
    hylo.get_address(address_field),
    new_address,
    ErrorCode::AdminNoop
  );
  self.address_field = address_field;
  self.new_address = new_address;
  self.proposal_time = clock.unix_timestamp;
  self.ttl_secs = validate_ttl(ttl_secs)?;
```

```
  self.approved = false;
  Ok(())
}
```

```
/// hylo.rs L143-L154
pub fn set_address(
  &mut self,
  address_field: AddressField,
  new_address: Pubkey,
) -> Result<AcceptAddressUpdateEvent> {
  let old_address = self.get_address(address_field);
  require_neq!(old_address, new_address, ErrorCode::AdminNoop);
  match address_field {
    AddressField::Admin => self.admin = new_address,
    AddressField::Treasury => self.treasury = new_address,
    AddressField::PauseAuthority => self.pause_authority = new_address,
  }
```

```
/// register_exo.rs L43-L50
#[account(
  init,
  payer = admin,
  space = 8 + ExoPair::INIT_SPACE,
  seeds = [EXO_PAIR, collateral_mint.key().as_ref()],
  bump,
)]
pub exo_pair: Account<'info, ExoPair>,
```

Mitigation Suggestion

When `address_field == AddressField::Admin`, validate that the proposed new admin can safely act as the payer for future admin instructions. For example, require the new admin account to be system-owned, or separate the admin authority from the payer account.

```
if address_field == AddressField::Admin {
  require_keys_eq!(
    new_admin_account.owner,
    system_program.key(),
    ErrorCode::InvalidAdminAddress
  );
}
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/4b3a4a1b567072602b5d140b5bd2192fb5d509c1.

ACC-L11 Active Stable-to-Lever Path During Upgrade Can Leave Earn Pool With xSOL and Overmint sHYUSD

LOW FIXED

Description

The earn-pool deposit path prices sHYUSD using only the HYUSD pool balance. It does not account for any xSOL held by the earn-pool.

During the upgrade window, if the `HYUSD -> XSOL` path remains enabled, a malicious user may use the rebalance/conversion path to move value into xSOL before the new earn-pool logic is active. After the upgrade, new deposits only see the HYUSD balance, so sHYUSD can be minted at an incorrect price.

Location

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/instructions/user_deposit.rs#L110-L118

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/instructions/initialize_earn_pool.rs#L42-L56

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-router/src/instructions/route.rs#L88-L95

Relevant Code

```
/// user_deposit.rs L110-L118
let lp_token_nav = lp_token_nav(
  UFix64::new(stablecoin_pool.amount),
  UFix64::new(lp_token_mint.supply),
)?;

let stablecoin_deposited = UFix64::new(amount_stablecoin);
let lp_token_amount = lp_token_out(stablecoin_deposited, lp_token_nav)?;
```

```
/// initialize_earn_pool.rs L42-L56
#[account(
  init,
  payer = admin,
  associated_token::mint = stablecoin_mint,
  associated_token::authority = pool_auth,
)]
pub stablecoin_pool: Account<'info, TokenAccount>,

#[account(
  init,
  payer = admin,
  associated_token::mint = levercoin_mint,
  associated_token::authority = pool_auth,
)]
pub levercoin_pool: Account<'info, TokenAccount>,
```

```
/// route.rs L88-L95
(HYUSD::MINT, XSOL::MINT) => (
  hylo_exchange::ID,
  ConvertStableToLeverLst {
    amount_stablecoin: amount,
    slippage_config,
  }
  .data(),
),
```

Mitigation Suggestion

Use a two-step upgrade process. First disable the `stable_to_lever` path so users cannot move HYUSD value into xSOL during the upgrade window. After that, upgrade the earn-pool program.

Remediation

Initially fixed by removing rebalance_stable_to_lever in https://github.com/hylo-so/protocol/pull/304/commits and levercoin deprecation in https://github.com/hylo-so/protocol/commit/637045c7db54ce071d92c9c189ad0ae88e189f15.

However, instead hylo will deploy the new earn pool as a separate program and allow legacy users to withdraw over time with no fees.

ACC-L12 Slippage Is Silently Ignored for sHYUSD Deposit and Withdraw Routes

LOW FIXED

Description

The router accepts `slippage_config` for every route, but the earn-pool deposit and withdraw instructions do not support slippage fields.

For `(HYUSD, SHYUSD)` and `(SHYUSD, HYUSD)`, the router silently discards the user-provided slippage config. A user or integration may believe they are protected, while the actual deposit or withdrawal executes with no minimum LP-out or minimum HYUSD-out check.

sHYUSD NAV can shift between sign and execution — yield harvest, absorb_loss, rebalance PnL, or changing a fee can all move the share price within the same transaction.

Location

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-router/src/instructions/route.rs#L48-L53

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-router/src/instructions/route.rs#L208-L221

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/lib.rs#L47-L63

Relevant Code

```
/// route.rs L48-L53
fn resolve_route(
  token_a: Pubkey,
  token_b: Pubkey,
  amount: u64,
  slippage_config: Option<SlippageConfig>,
  accounts: &[AccountInfo<'_>],
) -> Result<Instruction> {
```

```
/// route.rs L208-L221
(HYUSD::MINT, SHYUSD::MINT) => (
  hylo_earn_pool::ID,
  UserDeposit {
    amount_stablecoin: amount,
  }
  .data(),
),
(SHYUSD::MINT, HYUSD::MINT) => (
  hylo_earn_pool::ID,
  UserWithdraw {
    amount_lp_token: amount,
  }
  .data(),
),
```

```
/// lib.rs L47-L63
pub fn user_deposit<'me: 'info, 'info>(
  ctx: Context<UserDeposit<'info>>,
  amount_stablecoin: u64,
) -> Result<UserDepositEvent> {
  let event = UserDeposit::handle(ctx.accounts, amount_stablecoin)?;
  emit_cpi!(event);
  Ok(event)
```

```
}

pub fn user_withdraw<'me: 'info, 'info>(
  ctx: Context<UserWithdraw<'info>>,
  amount_lp_token: u64,
) -> Result<UserWithdrawEvent> {
  let event = UserWithdraw::handle(ctx.accounts, amount_lp_token)?;
  emit_cpi!(event);
  Ok(event)
}
```

Mitigation Suggestion

Either add slippage protection to earn-pool deposits and withdrawals, or make the router reject `Some(slippage_config)` for these routes.

```
(HYUSD::MINT, SHYUSD::MINT) | (SHYUSD::MINT, HYUSD::MINT) => {
  require!(slippage_config.is_none(), ErrorCode::UnsupportedSlippage);
  // build earn-pool instruction
}
```

A better fix is to add `min_lp_token_out` for deposits and `min_stablecoin_out` for withdrawals.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/1715c1724ef255ddc630db5e3accc7cd449b3bdd.

ACC-L13 Collapsed SwapLstToLst Route Can Execute the Swap in the Wrong Direction

LOW FIXED

Description

`route.rs` maps both `JITOSOL -> HYLOSOL` and `HYLOSOL -> JITOSOL` to the same `SwapLstToLst` instruction with `amount_lst_a: amount`.

The actual input token is determined by the `lst_a` accounts in `remaining_accounts`, not by the `token_a` argument passed to the router. If the caller passes `route(JITOSOL, HYLOSOL, amount, ...)` but provides HYLOSOL accounts in the `lst_a` slots, the router can execute the opposite swap direction. With loose or missing slippage protection, this can cause user loss.

Location

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-router/src/instructions/route.rs#L104-L111

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-exchange/src/instructions/swap_lst_to_lst.rs#L26-L65

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-exchange/src/instructions/swap_lst_to_lst.rs#L146-L158

Relevant Code

```
/// route.rs L104-L111
(JITOSOL::MINT, HYLOSOL::MINT) | (HYLOSOL::MINT, JITOSOL::MINT) => (
  hylo_exchange::ID,
  SwapLstToLst {
    amount_lst_a: amount,
    slippage_config,
  }
  .data(),
),
```

```
/// swap_lst_to_lst.rs L26-L65
// LST A
pub lst_a_mint: Box<Account<'info, Mint>>,

#[account(
  mut,
  token::mint = lst_a_mint,
  token::authority = user,
)]
pub lst_a_user_ta: Box<Account<'info, TokenAccount>>,

// LST B
pub lst_b_mint: Box<Account<'info, Mint>>,

#[account(
  mut,
  token::mint = lst_b_mint,
  token::authority = user,
)]
pub lst_b_user_ta: Box<Account<'info, TokenAccount>>,
```

```
/// swap_lst_to_lst.rs L146-L158
let lst_a_in = UFix64::<N9>::new(amount_lst_a);
let FeeExtract {
  fees_extracted,
```

```
  amount_remaining,
} = hylo.lst_swap_config()?.apply_fee(lst_a_in)?;

let lst_b_out = lst_a_header.price_sol.convert_lst_amount(
  epoch,
  amount_remaining,
  &lst_b_header.price_sol,
)?;
```

Mitigation Suggestion

Do not collapse both swap directions into one match arm, or verify that `token_a` and `token_b` match the actual `lst_a_mint` and `lst_b_mint` accounts before invoking.

```
require_keys_eq!(
  remaining_accounts[LST_A_MINT_INDEX].key(),
  token_a,
  ErrorCode::InvalidRouteAccounts
);

require_keys_eq!(
  remaining_accounts[LST_B_MINT_INDEX].key(),
  token_b,
  ErrorCode::InvalidRouteAccounts
);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/a241164234dc1acf973592d1417ff1662a9cb69d.

ACC-L14 Borrow Rate Cap Can Double After Reduced Solana Epoch Duration

LOW FIXED

Description

`MAX_RATE` is defined as a per-epoch borrow rate and the comment assumes about 182 epochs per year. This matches the current ~48 hour epoch duration.

SIMD-0525 reduces target slot time in stages while keeping epochs fixed at 432,000 slots. At the final 200ms slot target, epochs become about 24 hours, or roughly 365 epochs per year. If the same per-epoch cap is kept, the effective annualized borrow rate cap roughly doubles.

This can make the borrow-rate configuration allow a much higher annual rate than intended after the slot-time change.

Location

- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/borrow_rate.rs#L25-L26
- https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/borrow_rate.rs#L76-L85
- https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L212-L221
- https://github.com/solana-foundation/solana-improvement-documents/blob/main/proposals/0525-reduce-slot-times.md#timing-values

Relevant Code

```
/// borrow_rate.rs L25-L26
/// Maximum per-epoch rate (~10% annualized at 182 epochs/year)
const MAX_RATE: UFix64<N9> = UFix64::constant(600_000);
```

```
/// borrow_rate.rs L76-L85
pub fn validate(&self) -> Result<BorrowRateConfig> {
  let rate = self.rate()?;
  let fee = self.fee()?;
  if rate > UFix64::zero()
    && rate <= MAX_RATE
    && fee > UFix64::zero()
    && fee <= MAX_FEE
  {
    Ok(*self)
  } else {
```

```
/// harvest_borrow_rate.rs L212-L221
let stablecoin_to_harvest = {
  let raw_stablecoin = exo_pair
    .borrow_rate_config
    .apply_borrow_rate(levercoin_market_cap)?
    .checked_convert()
    .ok_or(ErrorCode::TokenAmountPrecisionError)?;
  exchange_context
    .validate_stablecoin_amount(raw_stablecoin)
    .or_else(|_| exchange_context.max_mintable_stablecoin())?
};
```

Mitigation Suggestion

The config rate and the cap must be updated accordingly compared to https://github.com/solana-foundation/solana-improvement-documents/blob/main/proposals/0525-reduce-slot-times.md#timing-values, as this will not be done in a single step.

Remediation

Fixed in https://github.com/hylo-so/sdk/pull/89 .

ACC-L15 Wrong Levercoin Mint PDA Prevents Pool Deprecation

LOW FIXED

Description

We found that `deprecate_levercoin_pool` validates `levercoin_mint` using the local program as the PDA derivation program.

However, the levercoin mint is created by the `hylo_exchange` program. Because `seeds::program = hylo_exchange::ID` is missing, Anchor derives a different PDA than the real levercoin mint. As a result, the instruction cannot validate the correct mint and may fail before closing the intended `levercoin_pool` account.

Location

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/instructions/deprecate_levercoin_pool.rs#L36-L40

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-exchange/src/instructions/initialize_mints.rs#L50-L59

Relevant Code

```
/// deprecate_levercoin_pool.rs L36-L40
#[account(
  seeds = [XSOL],
  bump = hylo.levercoin_mint_bump,
)]
pub levercoin_mint: Account<'info, Mint>,
```

```
/// initialize_mints.rs L50-L59
#[account(
  init,
  payer = admin,
  seeds = [XSOL],
  bump,
  mint::authority = levercoin_auth,
  mint::decimals = 6,
  mint::token_program = token_program,
)]
pub levercoin_mint: Account<'info, Mint>,
```

Mitigation Suggestion

Derive `levercoin_mint` with the `hylo_exchange` program ID, since that program owns the PDA derivation for the mint.

```
#[account(
  seeds = [XSOL],
  seeds::program = hylo_exchange::ID,
  bump = hylo.levercoin_mint_bump,
)]
pub levercoin_mint: Account<'info, Mint>,
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/5730b8f4909754eb67d692753a147298bfa059eb.

ACC-L16 Registering a new LST before harvest_yield blocks yield harvest for the epoch

LOW FIXED

Description

Registering a new LST via `register_lst` before `harvest_yield` has been called will block the yield harvest for the remainder of that epoch when the protocol is in Normal or Mode1 stability mode.

Root cause: `LstHeader::init()` sets `last_yield_harvest_epoch = current_epoch()` (line 272 in `lst_registry.rs`). When `harvest_yield` is subsequently called, `total_sol_to_harvest()` iterates over **all** LSTs in the registry and requires that each one satisfies `last_yield_harvest_epoch < current_epoch`. The newly registered LST has `last_yield_harvest_epoch == current_epoch`, so this check fails and the function returns `YieldHarvestEpoch` error, aborting the entire harvest.

Detailed flow:

1. Epoch `E` begins. `harvest_yield` has not yet been called.

2. Admin calls `register_lst` for a new LST. `LstHeader::init()` sets `last_yield_harvest_epoch = E`.

3. A crank (or anyone) calls `harvest_yield`. The global `yield_harvest_cache.is_stale(E)` check passes (harvest hasn't run yet this epoch).

4. In `commit_normal_harvest`, `total_sol_to_harvest()` iterates all LSTs. For the newly registered LST: `E < E` is `false`, so it enters the `else` branch and returns `Err(YieldHarvestEpoch)`.

5. The entire `harvest_yield` transaction reverts.

Downstream impact: Every user-facing operation — `mint_stablecoin`, `mint_levercoin`, `redeem_stablecoin`, `redeem_levercoin`, `swap_stable_to_lever`, `swap_lever_to_stable`, `swap_lst`, and `swap_lst_usdc` — requires `yield_harvest_cache.epoch == current_epoch` (checked via `ErrorCode::YieldHarvestHasNotRun`). Since `harvest_yield` cannot complete, **all protocol operations are blocked for the remainder of the epoch** (~2 days on mainnet).

Note: The Mode2/Depeg path (`commit_noop_harvest`) does not call `total_sol_to_harvest()`, so this issue only manifests when the protocol is in Normal or Mode1 stability mode — the most common operating states.

While `register_lst` is admin-gated (`has_one = admin`), this is still a significant operational risk. An admin performing a routine LST onboarding could inadvertently freeze the entire protocol. Additionally, if the admin key is compromised, an attacker could exploit this to deny service cheaply by registering a new LST at the start of each epoch before the harvest crank runs.

Location

https://github.com/hylo-so/protocol/blob/c231584b625bf904b71e8d1b1b7bba4d0867f446/programs/hylo-exchange/src/instructions/register_lst.rs#L0

Relevant Code

```
// lst_registry.rs:272 — LstHeader::init sets last_yield_harvest_epoch to current epoch

self.last_yield_harvest_epoch = self.price_sol.epoch;


// lst_registry.rs:158-181 — total_sol_to_harvest fails if any LST already has current epoch

pub fn total_sol_to_harvest(&mut self) -> Result<UFix64<N9>> {

  self
```

```
    .registry
    .iter_mut()
    .try_fold(UFix64::zero(), |acc, lst| {
      let current_epoch = current_epoch()?;
      if lst.lst_header.last_yield_harvest_epoch < current_epoch {
        lst.lst_header.last_yield_harvest_epoch = current_epoch;
        lst.lst_header.exit(&HyloExchange::id())?;
        let appreciation = lst.lst_header.sol_price_appreciation()?;
        let amount_lst = UFix64::<N9>::new(lst.vault.amount);
        let amount_sol = appreciation
          .mul_div_floor(amount_lst, UFix64::one())
          .ok_or(LstSolAppreciation)?;
        acc
          .checked_add(&amount_sol)
          .ok_or(LstAdditionOverflow.into())
      } else {
        Err(YieldHarvestEpoch.into()) // <-- newly registered LST hits this branch
      }
    })
}
```

Mitigation Suggestion

Two complementary approaches:

Option A — Skip newly registered LSTs during harvest: In `total_sol_to_harvest()`, instead of returning an error when `last_yield_harvest_epoch >= current_epoch`, skip the LST and continue accumulating from others. A newly registered LST with zero vault balance has no yield to harvest anyway:

```
if lst.lst_header.last_yield_harvest_epoch < current_epoch {
  // ... existing harvest logic ...
} else {
  // Already harvested or newly registered — skip, no yield to collect
  Ok(acc)
}
```

Option B — Set initial epoch to previous epoch: In `LstHeader::init()`, set `last_yield_harvest_epoch` to `current_epoch() - 1` (with a saturating subtraction for epoch 0). This allows the new LST to pass the `< current_epoch` check. Since the vault is freshly created with zero balance, the harvest computation for it would yield zero SOL anyway:

```
self.last_yield_harvest_epoch = self.price_sol.epoch.saturating_sub(1);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/pull/286/.

ACC-L17 Missing Minimum-Output Check Can Cause Silent Donation

LOW FIXED

Description

We found that neither UserDeposit nor UserWithdraw enforces a minimum non-zero output invariant. Due to integer rounding in share/NAV math: • A deposit can transfer stablecoin into the pool but mint 0 LP tokens. • A withdrawal can burn LP tokens but redeem 0 tokens out.

This creates a silent loss for users where small operations effectively donate value to the pool. In particular, deposits that mint 0 shares inflate the value of existing LP shares.

Location

https://github.com/hylo-so/protocol/blob/760b187a80a1af128bce3ab3e36a1911d9dc2496/programs/hylo-stability-pool/src/instructions/user_deposit.rs#L140

Relevant Code

Mitigation Suggestion

Require minted shares and redeemed tokens to be greater than zero, otherwise revert the instruction.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/52b27ef796665c43f5f30e0d8b5c6d5fb6315050.

ACC-L18 initialize_lst_registry_calculators is not idempotent and can corrupt the registry LUT on repeated calls

LOW FIXED

Description

We found that InitializeLstRegistryCalculators::handle() always extends the LUT with the 16 calculator accounts and has no guard to prevent running more than once. Re-running it appends an extra “preamble” block that will later be interpreted as LST blocks, causing LST registry parsing to fail and disabling LST price/yield workflows.

Location

https://github.com/hylo-so/protocol/blob/760b187a80a1af128bce3ab3e36a1911d9dc2496/programs/hylo-exchange/src/instructions/initialize_lst_registry_calculators.rs#L38

Relevant Code

Mitigation Suggestion

You can disable the instruction because it's already called.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/85d73f90b763f00d683282282bc0aacd50745263.

ACC-L19 Redeem Event Emits Incorrect Price Field

LOW FIXED

Description

We found that in redeem_levercoin.rs, the emitted event records sol_usd_price.lower as the redemption price. However, the actual LST amount returned to the user is calculated using sol_usd_price.upper (which is correctly used for the redeem direction in Conversion::token_to_lst).

This creates a mismatch between the price recorded in the event and the price actually used in the calculation, which can mislead monitoring systems, analytics, or accounting tools that rely on emitted events.

Location

https://github.com/hylo-so/protocol/blob/c231584b625bf904b71e8d1b1b7bba4d0867f446/programs/hylo-exchange/src/instructions/redeem_levercoin.rs#L210

Relevant Code

```
sol_usd_price: exchange_context.sol_usd_price.lower.into(),
```

Mitigation Suggestion

Emit sol_usd_price.upper in the redemption event so the recorded price matches the one used in the calculation.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/8e63bfbd3eba7c8e40ebec04a3507209c5f3126e.

ACC-L20 Stale Oracle Data Can Freeze Stability Pool Withdrawals

LOW FIXED

Description

We found that stale oracle data can block withdrawals even though the withdrawal amounts are calculated from pool balances first.

In user_withdraw, the program computes stablecoin_to_withdraw and levercoin_to_withdraw before loading LstExchangeContext. However, it still later requires the oracle-backed context only to emit NAV values and apply event-side pricing. Because of this, a stale SOL/USD oracle can freeze all LP withdrawals, even when the actual withdrawal amounts are already known and do not depend on fresh oracle data.

Location

https://github.com/hylo-so/protocol/blob/c231584b625bf904b71e8d1b1b7bba4d0867f446/programs/hylo-stability-pool/src/instructions/user_withdraw.rs#L182-L183

Relevant Code

```
let stablecoin_nav = exchange_context.stablecoin_nav()?;
    let levercoin_nav = exchange_context.levercoin_mint_nav()?;
```

Mitigation Suggestion

Remove the exchange_context loading process.

Remediation

Fixed in b2c624e6c2fea2391a90df12a7ab4846dc08a482.

ACC-L21 Stability pool PoolConfig setters allow no-op updates

LOW FIXED

Description

We found that `PoolConfig::update_admin` and `PoolConfig::update_withdrawal_fee` do not check that the new value differs from the old one, allowing pointless state writes and event emissions.

Location

https://github.com/hylo-so/protocol/blob/ed3ac2ccf227a2a90f4a5988eef15d24f26cb4f5/programs/hylo-stability-pool/src/state/pool_config.rs#L36-L59

Relevant Code

```
pub fn update_admin(
    &mut self,
    new_admin: Pubkey,
  ) -> Result<UpdateAdminEvent> {
    let old_admin = self.admin;
    self.admin = new_admin;
    Ok(UpdateAdminEvent {
      old_admin,
      new_admin,
    })
  }


  pub fn update_withdrawal_fee(
    &mut self,
    new_withdrawal_fee: UFixValue64,
  ) -> Result<UpdateWithdrawalFeeEvent> {
    Self::validate_withdrawal_fee(new_withdrawal_fee)?;
    let old_withdrawal_fee = self.withdrawal_fee;
    self.withdrawal_fee = new_withdrawal_fee;
    Ok(UpdateWithdrawalFeeEvent {
      old_withdrawal_fee,
      new_withdrawal_fee,
    })
  }
```

Mitigation Suggestion

Validate for pointless updates. Similar check: https://github.com/hylo-so/protocol/blob/c231584b625bf904b71e8d1b1b7bba4d0867f446/programs/hylo-exchange/src/state/hylo.rs#L198-L201

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/7fca02352854bc06e690ac5db2399364c248dbe0.

ACC-L22 Hylo LST swap fee update stores unvalidated values

LOW FIXED

Description

We found that `Hylo::update_lst_swap_fee` assigns `new_lst_swap_fee` without validating its exponent/range.

Location

https://github.com/hylo-so/protocol/blob/c231584b625bf904b71e8d1b1b7bba4d0867f446/programs/hylo-exchange/src/state/hylo.rs#L257-L268

Relevant Code

```
pub fn update_lst_swap_fee(
    &mut self,
    new_lst_swap_fee: UFixValue64,
  ) -> Result<UpdateSwapFeeEvent> {
    require!(self.lst_swap_fee != new_lst_swap_fee, ErrorCode::AdminNoop);
    let old_lst_swap_fee = self.lst_swap_fee;
    self.lst_swap_fee = new_lst_swap_fee;
    Ok(UpdateSwapFeeEvent {
      old_swap_fee: old_lst_swap_fee,
      new_swap_fee: new_lst_swap_fee,
    })
  }
```

Mitigation Suggestion

Validate the fee.

Similar validation: https://github.com/hylo-so/protocol/blob/c231584b625bf904b71e8d1b1b7bba4d0867f446/programs/hylo-exchange/src/state/exo_pair.rs#L194

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/8122ae70916aabb17fe85851cd27f1b806b8db65.

ACC-L23 create_metadata Incorrectly Passes Mint as Rent and System Program

LOW FIXED

Description

We found that token::create_metadata incorrectly passes the mint account in place of the rent sysvar and system program accounts.

Location

https://github.com/hylo-so/protocol/blob/e539bdd3f40438a4152a0fa918f1982f4205c9c9/programs/hylo-exchange/src/token.rs#L158-L159

Relevant Code

```
system_program: mint.to_account_info(),
    rent: mint.to_account_info(),
```

Mitigation Suggestion

Pass the correct Rent sysvar and System Program accounts to create_metadata instead of the mint account.

Remediation

Fixed in https://github.com/hylo-so/protocol/pull/272.

ACC-L24 Shared PDA Seeds Can Collide and Merge Vault Control Across Pairs

LOW FIXED

Description

We found that LST, EXO, and USDC pairs derive vault/fee authorities using the same seed pattern: [VAULT_AUTH, mint.key().as_ref()].

This is dangerous because if the same mint is used in multiple roles (e.g., as both an LST mint and an EXO mint), the PDAs will collide. That means both pairs would share the same vault/fee authority and effectively control the same vault domain, leading to cross-pair fund mixing and broken access boundaries.

Location

https://github.com/hylo-so/protocol/blob/e539bdd3f40438a4152a0fa918f1982f4205c9c9/programs/hylo-exchange/src/instructions/register_lst.rs#L35-L47

https://github.com/hylo-so/protocol/blob/e539bdd3f40438a4152a0fa918f1982f4205c9c9/programs/hylo-exchange/src/instructions/register_exo.rs#L47-L59

https://github.com/hylo-so/protocol/blob/e539bdd3f40438a4152a0fa918f1982f4205c9c9/programs/hylo-exchange/src/instructions/initialize_usdc.rs#L35-L47

Relevant Code

Mitigation Suggestion

Include the pair type/domain (e.g., LST, EXO, USDC) in the PDA seeds so each vault/fee authority is unique even if the mint is the same.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/6df5a67a85f53f835c753205829a9c23421a971c.

ACC-L25 Front-Running Can Desync HYLO Virtual Stablecoin From HYUSD Supply

LOW ACKNOWLEDGED

Description

Multiple virtual stablecoins exist (e.g., usdc_pair.virtual_stablecoin, exo_pair.virtual_stablecoin, hylo.virtual_stablecoin). All of them mint HYUSD, but HYLO's virtual stablecoin must stay 1:1 with HYUSD minted by SOL LST (lever/stable path).

The problem is that InitializeLstVirtualStablecoin is front-runnable. An attacker can front-run initialization and mint HYUSD through other paths (e.g., sSwapStablecoinUsdc), which increases: • HYUSD total supply, and • usdc_pair.virtual_stablecoin

But as we call this before InitializeLstVirtualStablecoin and it increases HYUSD total supply, hylo.virtual_stablecoin will be depegged from HYUSD minted by SOL LST (lever/stable path). This creates a mismatch where HYLO's virtual stablecoin no longer matches the intended HYUSD supply baseline for LST-minted HYUSD.

But the attack surface is limited due to the fact that this is only possible if the admin register exo pair or the usc pair before InitializeLstVirtualStablecoin.

Location

https://github.com/hylo-so/protocol/blob/init-v2/programs/hylo-exchange/src/instructions/initialize_lst_virtual_stablecoin.rs

Relevant Code

Mitigation Suggestion

Before registering any exo or USDC pair, ensure that InitializeLstVirtualStablecoin is called.

Remediation

Acknowledged.

ACC-I1 Same-Epoch Stale LST Prices Can Misprice LST Swaps and Leak Vault Value

INFO ACKNOWLEDGED

Description

`swap_lst_to_lst` quotes the output amount using the cached `LstHeader.price_sol` values for both LSTs. The only freshness requirement is that the cached prices are stamped with the current epoch.

However, `LstHeader::update_price` ignores price changes within the same epoch. If an LST price changes after the first price update in an epoch, the cached value can stay stale until the next epoch.

An attacker can use this stale price window to swap an overvalued cached input LST or withdraw an undervalued cached output LST. After the token transfers, `refresh_lst_vault` updates the SOL cache using the same stale prices, so the vault delta is booked at the stale value. No PnL settlement corrects the loss.

Location

• https://github.com/hylo-so/protocol/blob/6f53f619e61b009acbe3d631a7d463b0c5544e0f/programs/hylo-exchange/src/instructions/swap_lst_to_lst.rs#L151-L214

• https://github.com/hylo-so/protocol/blob/6f53f619e61b009acbe3d631a7d463b0c5544e0f/programs/hylo-exchange/src/state/lst_registry.rs#L314-L329

• https://github.com/hylo-so/protocol/blob/6f53f619e61b009acbe3d631a7d463b0c5544e0f/programs/hylo-exchange/src/state/hylo.rs#L355-L367

• https://github.com/hylo-so/protocol/blob/6f53f619e61b009acbe3d631a7d463b0c5544e0f/programs/hylo-exchange/src/instructions/update_lst_prices.rs#L37-L42

Relevant Code

```
let lst_b_out = lst_a_header.price_sol.convert_lst_amount(
  epoch,
  amount_remaining,
  &lst_b_header.price_sol,
)?;
```

```
hylo.refresh_lst_vault(
  &lst_a_header.price_sol,
  lst_a_amount_before,
  UFix64::<N9>::new(lst_a_vault.amount),
  epoch,
)?;
hylo.refresh_lst_vault(
  &lst_b_header.price_sol,
  lst_b_amount_before,
  UFix64::<N9>::new(lst_b_vault.amount),
  epoch,
)?;
```

```
pub fn update_price(
  &mut self,
  new_price: UFix64<N9>,
  this_epoch: u64,
) -> Result<Self> {
  match this_epoch.cmp(&self.price_sol.epoch) {
    Ordering::Greater => {
      self.prev_price_sol = self.price_sol;
      self.price_sol = LstSolPrice::new(new_price.into(), this_epoch);
      Ok(*self)
    }
```

```
    Ordering::Equal => Ok(*self),
    Ordering::Less => Err(LstPriceEpochsInvalid.into()),
  }
}
```

```
let sol_before = price_sol.convert_lst_to_sol(amount_before, epoch)?;
let sol_after = price_sol.convert_lst_to_sol(amount_after, epoch)?;
self.total_sol_cache.decrement(sol_before, epoch)?;
self.total_sol_cache.increment(sol_after, epoch)?;
```

```
let updated_mints = registry.update_all_lst_sol_prices()?;
let new_total_sol = registry.total_sol()?;
hylo.total_sol_cache.set(new_total_sol, current_epoch()?)?;
```

Mitigation Suggestion

Do not price `swap_lst_to_lst` only from epoch-fresh cached prices. Validate both LST prices against live Sanctum calculator values before quoting and before refreshing the vault cache.

Remediation

This issue has been acknowledged due to low intra-epoch value differentials.

ACC-I2 Repeated Same-Epoch LST Price Updates Can Reduce Harvested Yield

INFO FIXED

Description

`update_lst_prices` can be called more than once in the same epoch. When it is called in the same epoch, `LstHeader::update_price` overwrites `price_sol` in place and does not preserve the earlier price for yield harvesting.

Yield harvesting later computes appreciation as `price_sol - prev_price_sol`. If someone calls `update_lst_prices` near the end of an epoch, `price_sol` is moved forward before the next harvest. The next epoch's harvest then only captures the price increase after that late update, and the earlier appreciation from the epoch is not harvested.

For example:

```
Epoch 2 start price = 1.50
Epoch 3 start price = 1.60
Harvest delta = 0.10
```

But if `update_lst_prices` is called again at the end of epoch 2:

```
Epoch 2 start price = 1.50
Epoch 2 end price = 1.52
Epoch 3 start price = 1.60
Harvest delta = 0.08
```

The `0.02` appreciation is skipped by the harvest calculation.

Location

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/update_lst_prices.rs#L37-L45

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L135-L151

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L301-L319

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L171-L195

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L342-L345

• https://github.com/hylo-so/protocol/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_yield.rs#L168-L181

Relevant Code

```
let updated_mints = registry.update_all_lst_sol_prices()?;
let new_total_sol = registry.total_sol()?;
hylo.total_sol_cache.set(new_total_sol, current_epoch()?)?;
```

```
let new_price = ctx.lst_sol_price(&lst.mint, &lst.pool_state)?;
// Update in place
lst.lst_header.update_price(new_price, current_epoch)?;
```

```
match this_epoch.cmp(&self.price_sol.epoch) {
  Ordering::Greater => {
    self.prev_price_sol = self.price_sol;
    self.price_sol = LstSolPrice::new(new_price.into(), this_epoch);
    Ok(*self)
  }
  Ordering::Equal => {
    self.price_sol = LstSolPrice::new(new_price.into(), this_epoch);
    Ok(*self)
  }
```

```
let appreciation = lst.lst_header.sol_price_appreciation()?;
let amount_lst = UFix64::<N9>::new(lst.vault.amount);
let amount_sol = appreciation
  .mul_div_floor(amount_lst, UFix64::one())
  .ok_or(LstSolAppreciation)?;
```

```
pub fn sol_price_appreciation(&self) -> Result<UFix64<N9>> {
  self.price_sol.checked_delta(&self.prev_price_sol)
}
```

Mitigation Suggestion

Do not let same-epoch price updates overwrite the price used as the harvest baseline. For example, only update `price_sol` once per epoch, or track a separate `harvest_start_price` that remains fixed until yield is harvested.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/343646bb17be8f95ab7d7bc661e3401e48d867f0.

ACC-I3 Zero Earn Pool Balance Can Block New Deposits While LP Supply Remains

INFO FIXED

Description

If all stablecoin in the earn pool is burned while LP token supply is still positive, new deposits can fail.

`lp_token_nav` returns `1` only when LP supply is zero. If LP supply is nonzero and `stablecoin_pool.amount` is zero, the LP NAV becomes zero. `user_deposit` then passes that zero NAV into `lp_token_out`, which divides by the NAV and can fail due to division by zero.

This can happen after `absorb_loss` burns the full stablecoin pool balance. In that state, deposits cannot be used to restore the pool cleanly.

Location

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/earn_pool_math.rs#L16-L36

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/user_deposit.rs#L103-L121

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-earn-pool/src/instructions/absorb_loss.rs#L72-L88

Relevant Code

```
pub fn lp_token_nav(
  stablecoin_in_pool: UFix64<N6>,
  lp_token_supply: UFix64<N6>,
) -> Result<UFix64<N6>> {
  if lp_token_supply == UFix64::zero() {
    Ok(UFix64::one())
  } else {
    stablecoin_in_pool
      .mul_div_ceil(UFix64::one(), lp_token_supply)
      .ok_or(LpTokenNav.into())
  }
}
```

```
pub fn lp_token_out(
  amount_stablecoin_in: UFix64<N6>,
  lp_token_nav: UFix64<N6>,
) -> Result<UFix64<N6>> {
  amount_stablecoin_in
    .mul_div_floor(UFix64::one(), lp_token_nav)
    .ok_or(LpTokenOut.into())
}
```

```
let lp_token_nav = lp_token_nav(
  UFix64::new(stablecoin_pool.amount),
  UFix64::new(lp_token_mint.supply),
)?;

let stablecoin_deposited = UFix64::new(amount_stablecoin);
let lp_token_amount = lp_token_out(stablecoin_deposited, lp_token_nav)?;
```

```
let pool_balance = UFix64::<N6>::new(stablecoin_pool.amount);
let requested_loss = UFix64::<N6>::new(amount);
```

```
// Burn either entire pool or requested loss
let amount_stablecoin_burned = pool_balance.min(requested_loss);
```

Mitigation Suggestion

If the stablecoin pool balance reaches zero while LP supply is still nonzero, pause or explicitly disable new deposits until the state is recovered.

```
require!(
  stablecoin_pool.amount > 0 || lp_token_mint.supply == 0,
  ErrorCode::DepositDisabled
);
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/0a42d4606bbf6a34a234f202a445054335859566.

ACC-I4 100% Borrow Rate Fee Can Redirect All Exo Borrow Harvest Away From the Earn Pool

INFO FIXED

Description

`BorrowRateConfig` allows the admin to set the borrow-rate fee up to `100%`.

During borrow-rate harvest, the protocol first calculates the total stablecoin harvest, then splits it into `fees_extracted` and `amount_remaining`. Fees are minted to the treasury fee vault, while only `amount_remaining` is minted to the earn pool.

If the borrow-rate fee is set to `100%`, xASSET holders still pay the borrow rate, but none of the harvested amount flows to the earn pool or LPs. The full amount is routed to treasury.

This is inconsistent with the rest of the codebase, where configurable fees are capped much lower: yield harvest fees cap at `10%`, asset swap fees cap at `1%`, levercoin fees cap at `10%`, and LST rebalance fees cap at `0.5%`.

Location

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/borrow_rate.rs#L25-L82

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/exo_pair.rs#L230-L240

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/harvest_borrow_rate.rs#L224-L278

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/yields.rs#L10-L72

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/asset_swap_config.rs#L7-L33

• https://github.com/hylo-so/sdk/blob/708e7a3656014804867b9ef40bf30f743ca49f17/hylo-core/src/fees/controller.rs#L13-L49

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/state/lst_registry.rs#L252-L350

Relevant Code

```
/// Maximum fee exacted against borrow rate
const MAX_FEE: UFix64<N4> = UFix64::constant(10_000);

// ...

if rate > UFix64::zero()
  && rate <= MAX_RATE
  && fee > UFix64::zero()
  && fee <= MAX_FEE
{
  Ok(*self)
}
```

```
let FeeExtract {
  fees_extracted,
  amount_remaining,
} = exo_pair
  .borrow_rate_config
  .apply_fee(stablecoin_to_harvest)?;

// Update virtual stablecoin
exo_pair.virtual_stablecoin.mint(stablecoin_to_harvest)?;
```

```
// Send stablecoin fees to vault
mint_tokens(
  stablecoin_fee_vault,
  stablecoin_mint,
  stablecoin_auth,
  token_program,
  fees_extracted.bits,
  signer_seeds!(
    MINT_AUTH,
    stablecoin_mint.key(),
    hylo.stablecoin_auth_bump
  ),
)?;

// Send remaining stablecoin to earn pool
mint_tokens(
  stablecoin_pool,
  stablecoin_mint,
  stablecoin_auth,
  token_program,
  amount_remaining.bits,
```

```
/// 1000 bps (10%)
const MAX_FEE: UFix64<N4> = UFix64::constant(1000);
```

```
/// 100 bps (1%)
const MAX_FEE: UFix64<N4> = UFix64::constant(100);
```

```
const MAX_FEE: UFix64<N4> = UFix64::constant(1000);
```

```
/// 50 bps (0.5%)
const MAX_REBALANCE_FEE: UFix64<N5> = UFix64::constant(500);
```

Mitigation Suggestion

Cap the borrow-rate fee to a lower value, such as `10%`, to match the yield-harvest and levercoin fee convention.

```
/// 1000 bps (10%)
const MAX_FEE: UFix64<N4> = UFix64::constant(1000);
```

Remediation

Fixed in https://github.com/hylo-so/sdk/commit/b7a6b88ba14e3736e61d7651e4ce2c4bb2a1bba6.

ACC-I5 Misleading Self-CPI Comment Can Lead to Incorrect Rebalance Settlement Assumptions

INFO FIXED

Description

The comment says `SettleRebalancePnlExo` is a self-CPI, but the code only constructs the account struct and calls `.handle(rebalance_pnl)` directly.

This can lead reviewers and maintainers to assume there is a CPI boundary or fresh account deserialization, while the function is actually using the same in-memory account wrappers from the current instruction.

Location

• https://github.com/accretion-xyz/2026-hylo-protocol-v2-audit-A26HYL1/blob/4669205bc719e21b2126bfa553732ea1c68683ef/programs/hylo-exchange/src/instructions/swap_exo_usdc.rs#L459-L476

Relevant Code

```
// Self-CPI to settle `PnL`
SettleRebalancePnlExo {
  hylo,
  pool_config,
  exo_pair,
  stablecoin_mint_auth,
  vault_auth,
  pool_auth,
  settlement_auth,
  collateral_vault,
  stablecoin_pool,
  collateral_mint,
  stablecoin_mint,
  collateral_usd_pyth_feed,
  token_program,
  earn_pool,
}
.handle(rebalance_pnl)?;
```

Mitigation Suggestion

Update the comment to describe the actual behavior, or remove it.

```
// Settle `PnL`
SettleRebalancePnlExo {
  // ...
}
.handle(rebalance_pnl)?;
```

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/a9eaf4f8fac0ba815326790c6018c088b6588725.

ACC-I6 Unused Earn-Pool Error Codes

INFO FIXED

Description

The earn-pool defines `DepositDisabled` and `LstPairPaused`, but they are never used in any earn-pool instruction. These unused errors should be removed to keep the code clean and avoid confusion.

Location

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/error.rs#L4-L20

Relevant Code

```
/// error.rs L4-L20
#[error_code]
pub enum ErrorCode {
  #[msg("Deposits to pool disabled due to active rebalancing.")]
  DepositDisabled,
  ...
  #[msg("Trading for the LST pair has been paused by admin.")]
  LstPairPaused,
  ...
}
```

Mitigation Suggestion

Remove the unused `DepositDisabled` and `LstPairPaused` error variants.

Remediation

Fixed in https://github.com/hylo-so/protocol/commit/8e324e0adfaf0f19381f25e588e89eb04e65620a.

ACC-I7 First Depositor Can Inflate sHYUSD Share Price and Steal Later Deposits

INFO FIXED

Description

The deposit path is vulnerable to a first-depositor inflation attack when the LP token supply is zero.

When supply is zero, `lp_token_nav` returns `1`. The first depositor can deposit a very small amount and receive LP tokens at a 1:1 price. After that, the attacker can transfer HYUSD directly into `stablecoin_pool`, increasing the pool balance without minting new LP tokens. This inflates the LP token NAV.

A later user deposit then mints fewer LP tokens because `lp_token_out` uses floor division. The attacker can withdraw using their LP token and capture part of the victim's deposit through the inflated share price and rounding loss.

The current `ZeroLpDeposit` check only ensures that the victim receives at least one LP token. It does not prevent the attacker from capturing value.

This is most relevant when the pool is first initialized, or if the LP token supply ever returns to zero.

Location

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/instructions/user_deposit.rs#L110-L121

• https://github.com/hylo-so/protocol/blob/b2c624e6c2fea2391a90df12a7ab4846dc08a482/programs/hylo-earn-pool/src/instructions/user_withdraw.rs#L125-L133

• https://github.com/hylo-so/sdk/blob/3143d060502e979d1dc0dbe1aa4d528eaf9b9e47/hylo-core/src/earn_pool_math.rs#L16-L47

Relevant Code

```
/// user_deposit.rs L110-L121
let lp_token_nav = lp_token_nav(
  UFix64::new(stablecoin_pool.amount),
  UFix64::new(lp_token_mint.supply),
)?;

let stablecoin_deposited = UFix64::new(amount_stablecoin);
let lp_token_amount = lp_token_out(stablecoin_deposited, lp_token_nav)?;

require_gt!(lp_token_amount.bits, u64::MIN, ErrorCode::ZeroLpDeposit);
```

```
/// earn_pool_math.rs L16-L37
pub fn lp_token_nav(
  stablecoin_in_pool: UFix64<N6>,
  lp_token_supply: UFix64<N6>,
) -> Result<UFix64<N6>> {
  if lp_token_supply == UFix64::zero() {
    Ok(UFix64::one())
  } else {
    stablecoin_in_pool
      .mul_div_ceil(UFix64::one(), lp_token_supply)
      .ok_or(LpTokenNav.into())
  }
}

pub fn lp_token_out(
  amount_stablecoin_in: UFix64<N6>,
  lp_token_nav: UFix64<N6>,
) -> Result<UFix64<N6>> {
```

```
  amount_stablecoin_in
    .mul_div_floor(UFix64::one(), lp_token_nav)
    .ok_or(LpTokenOut.into())
}
```

```
/// user_withdraw.rs L125-L133
let lp_tokens_to_burn = UFix64::new(amount_lp_token);
let lp_token_supply = UFix64::new(lp_token_mint.supply);
let stablecoin_in_pool = UFix64::new(stablecoin_pool.amount);
let stablecoin_to_withdraw = amount_token_to_withdraw(
  lp_tokens_to_burn,
  lp_token_supply,
  stablecoin_in_pool,
)?;
```

Mitigation Suggestion

Permanently lock or burn a minimum amount of LP tokens to a dead address. This prevents the first depositor from controlling the full LP supply and makes direct donation attacks economically unprofitable.

Remediation

Fixed. https://solscan.io/tx/2H5RGFgHnewkhorqHVnWcHykVK6NSGvgK1JnmEtBqyMw3uuGdDkFFE7yatwvBJ5mCjuxB8a4LyJqBeAkEJJvJjHo

ACC-I8 Hardcoded 9-Decimal Assumption Can Misprice Non-9-Decimal LSTs

INFO ACKNOWLEDGED

Description

We found that SanctumContext::lst_sol_price hardcodes “1 token” as UFix64<N9>::one() (1e9 base units), which assumes every registered LST mint has 9 decimals. However, register_lst / LstHeader::init do not enforce lst_mint.decimals == 9.

As a result, if a non-9-decimal LST is ever registered, the program will compute the price of “1 token” incorrectly and propagate wrong pricing through the system. Even though Sanctum’s public list appears to use 9-decimal LSTs today, the list is external and archived, so relying on that assumption without on-chain validation is unsafe. ■

https://github.com/igneous-labs/sanctum-lst-list/blob/master/sanctum-lst-list.toml

Location

https://github.com/hylo-so/protocol/blob/760b187a80a1af128bce3ab3e36a1911d9dc2496/programs/hylo-exchange/src/instructions/register_lst.rs#L70

Relevant Code

Mitigation Suggestion

Enforce lst_mint.decimals == 9 during LST registration.

Remediation

Acknowledged.

ACC-I9 UpdateLstPricesEvent Can Overflow When More Than 10 LSTs Are Registered

INFO ACKNOWLEDGED

Description

We found that UpdateLstPricesEvent uses [max_len(10)] for updated_mints only to satisfy a compile-time size limit, but the program fills updated_mints with all registered LST mints. If more than 10 LSTs are registered, the returned data can exceed the expected size, which may cause truncated or invalid CPI return data, or event emission failure.

Location

https://github.com/hylo-so/protocol/blob/760b187a80a1af128bce3ab3e36a1911d9dc2496/programs/hylo-exchange/src/events.rs#L236-L242

Relevant Code

```
#[event]
#[derive(InitSpace)]
pub struct UpdateLstPricesEvent {
  #[max_len(10)]
  pub updated_mints: Vec<Pubkey>,
  pub new_total_sol: UFixValue64,
}
```

Mitigation Suggestion

Resize the event design so it safely supports all registered LSTs.

Remediation

Acknowledged, hylo doesn't intend to implement more than 10 LSTs in the protocol.

ACC-I10 New LP NAV Ignores Existing xSOL, Letting New Depositors Drain Pool

INFO FIXED

Description

We found a risk in the new “no volatility pushed to sHYUSD” design. After the upgrade, LP NAV is calculated only from **HYUSD**, while the program still holds a large amount of **xSOL** from the old version.

Because xSOL is no longer counted in NAV, the NAV becomes **too low**. This lets a new user deposit HYUSD and receive **more LP tokens than they should**. The user can then withdraw and receive their HYUSD (minus fees) **plus extra xSOL**, draining the pool over time.

Location

https://github.com/hylo-so/sdk/blob/2aaca46f396394d9eeae8adb6b0abb445337e7d2/hylo-core/src/stability_pool_math.rs#L9-L27

Relevant Code

```
/// Computes NAV for the stability pool's LP token.
///
///
///                   stablecoin_in_pool
/// lp_token_nav =  --------------------
///                     lp_token_supply
///
pub fn lp_token_nav(
  stablecoin_in_pool: UFix64<N6>,
  lp_token_supply: UFix64<N6>,
) -> Result<UFix64<N6>> {
  if lp_token_supply == UFix64::zero() {
    Ok(UFix64::one())
  } else {
    stablecoin_in_pool
      .mul_div_ceil(UFix64::one(), lp_token_supply)
      .ok_or(LpTokenNav.into())
  }
}
```

Mitigation Suggestion

Keep using the old LP NAV calculation (including xSOL + HYUSD) until the stability pool xSOL balance is fully removed, then switch to HYUSD-only NAV.

Remediation

The new earn pool will launch as a separate program and the old pool will migrate liquidity to it through incentives.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:
• Solana-specific vulnerabilities
• Access control issues
• Arithmetic errors and precision loss
• Race conditions and MEV opportunities
• Logic errors and edge cases
• Performance optimization opportunities
• Invariant violations
• Account confusion vulnerabilities
• Authority check omissions
• Token22 implementation risks and SPL-related pitfalls
• Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.