# LiquidRoyalty Audit Report

By AstraSec Team

December 2, 2025

# Scope

- https://github.com/stratosphere-network/LiquidRoyaltyContracts.git
- Commit id: b8234e5
- Contracts: BaseVault.sol, JuniorVault.sol, ReserveVault.sol, UnifiedSeniorVault.sol

# VN001: Unauthorized Withdrawal in BaseVault::_withdraw()

- Anyone can withdraw someone else's assets without approval.

```
function _withdraw(
    address caller,
    address receiver,
    address owner,
    uint256 assets,
    uint256 shares
) internal virtual override {
    // Calculate 1% withdrawal fee
    uint256 withdrawalFee = (assets * MathLib.WITHDRAWAL_FEE) / MathLib.
    uint256 netAssets = assets - withdrawalFee;

    // Iterative approach: Try up to 3 times to free up liquidity
    uint256 maxAttempts = 3;
    uint256 totalFreed = 0;
```

if (caller != owner) _spendAllowance(...)

# VN002: Lack of Input Validation in depositLP()

```
function depositLP(address lpToken, uint256 amount) external returns (uint256 depo
    if (lpToken == address(0)) revert ZeroAddress();
    if (amount == 0) revert InvalidAmount();
    if (address(kodiakHook) == address(0)) revert KodiakHookNotSet();

    // Transfer LP from user to hook
    IERC20(lpToken).safeTransferFrom(msg.sender, address(kodiakHook), amount);

    emit PendingLPDepositCreated(depositId, msg.sender, lpToken, amount, expiresAt
```

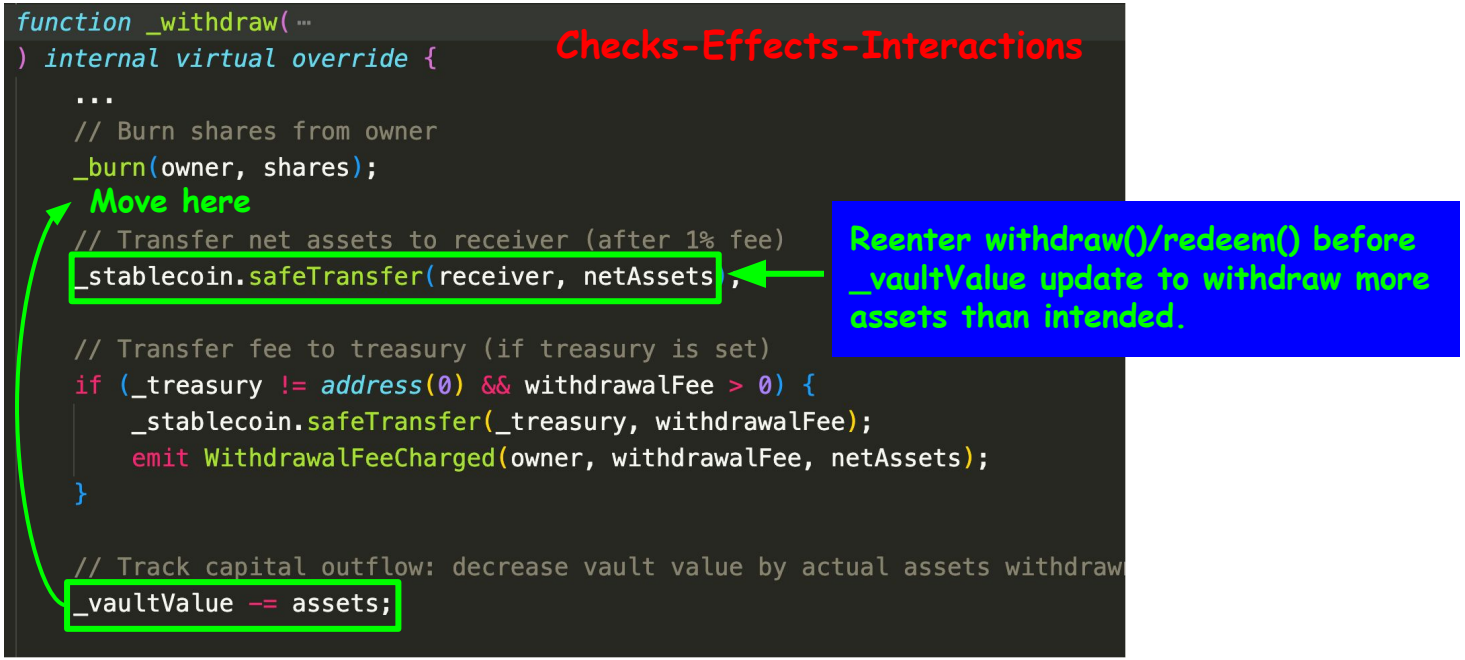
**Issues:**
1. Deposit **faked** lpToken, get real lpToken
2. If multiple LP tokens are whitelisted, how does kodiakHook know which one to return to the user?

- Affected: JuniorVault/UnifiedSeniorVault

```
function cancelPendingDeposit(uint256 depositId) external {
    PendingLPDeposit storage deposit = _pendingDeposits[depositId];

    if (deposit.depositor == address(0)) revert DepositNotFound();
    if (deposit.depositor != msg.sender) revert NotDepositor();
    if (deposit.status != DepositStatus.PENDING) revert DepositNotPending();

    // Transfer LP back from hook to depositor
    kodiakHook.transferIslandLP(deposit.depositor, deposit.amount);

    // Update status
    deposit.status = DepositStatus.CANCELLED;

    emit PendingLPDepositCancelled(depositId, deposit.depositor);
```

# VN003-1: Reentrancy Risk Caused by Breaking CEI Pattern

- Affected: BaseVault::_withdraw()



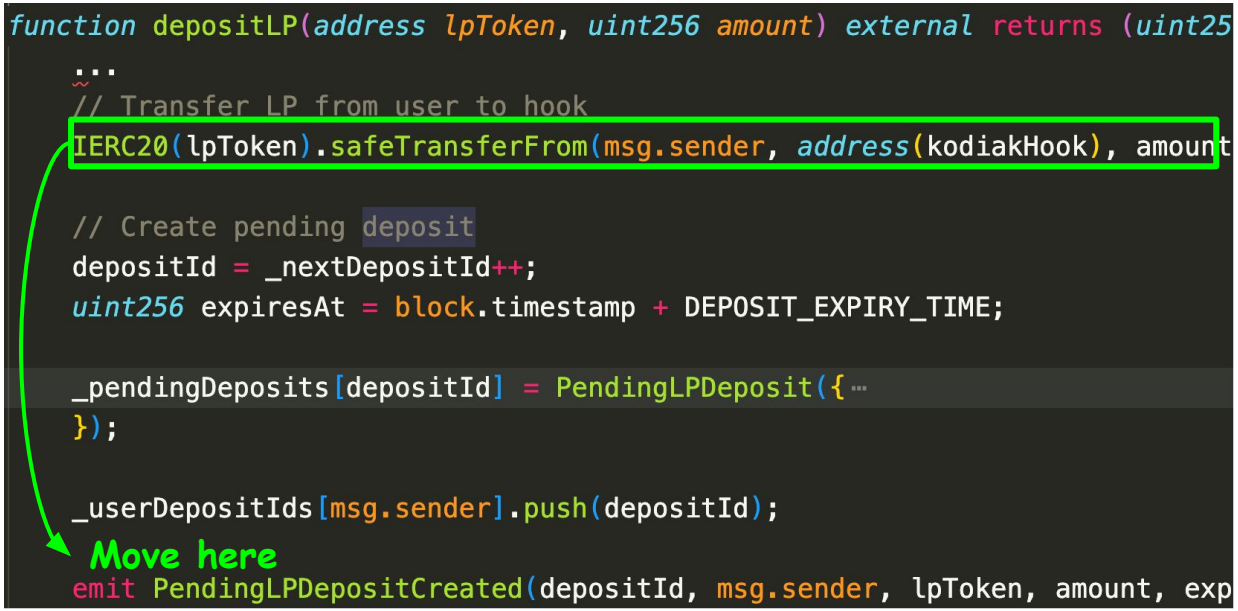
# VN003-2: Reentrancy Risk Caused by Breaking CEI Pattern

- Affected: UnifiedSeniorVault::withdraw()

```
function withdraw(uint256 amount, address receiver, address owner) public virt
    ...

    // Burn snrUSD
    _burn(owner, amount);
    Move here
    // Transfer net amount to receiver (after both early penalty and withdrawa
    _stablecoin.transfer(receiver, netAssets);

    // Transfer withdrawal fee to treasury
    if (_treasury != address(0) && withdrawalFee > 0) {…
    }

    // Track capital outflow: decrease vault value by amount after early penal
    _vaultValue -= amountAfterEarlyPenalty;
```

# VN003-3: Reentrancy Risk Caused by Breaking CEI Pattern

- Affected: JuniorVault::depositLP(), UnifiedSeniorVault::depositLP()



# VN003-4: Reentrancy Risk Caused by Breaking CEI Pattern

- Affected: JuniorVault::approveLPDeposit()/rejectLPDeposit()/cancelPendingDeposit()/claimExpiredDeposit(), UnifiedSeniorVault::approveLPDeposit()/rejectLPDeposit()/cancelPendingDeposit()/claimExpiredDeposit()

```
function rejectLPDeposit(uint256 depositId, string calldata reason) external
    PendingLPDeposit storage deposit = _pendingDeposits[depositId];

    if (deposit.depositor == address(0)) revert DepositNotFound();
    if (deposit.status != DepositStatus.PENDING) revert DepositNotPending();
    Move here
    // Transfer LP back from hook to depositor
    kodiakHook.transferIslandLP(deposit.depositor, deposit.amount);

    // Update status
    deposit.status = DepositStatus.REJECTED;
```

# VN004-1: Potential Sandwich/MEV Attacks in Vault Contracts

- Affected: BaseVault::updateVaultValue()/setVaultValue()

```
function updateVaultValue(int256 profitBps) public virtual onlyPriceFeedMa
    // Validate reasonable range
    if (profitBps < MIN_PROFIT_BPS || profitBps > MAX_PROFIT_BPS) {
        revert InvalidProfitRange();
    }

    uint256 oldValue = _vaultValue;

    // Apply profit/loss: V_new = V_old × (1 + profitBps / 10000)
    _vaultValue = MathLib.applyPercentage(oldValue, profitBps);
    _lastUpdateTime = block.timestamp;

    // Hook for derived contracts to execute post-update logic
    _afterValueUpdate(oldValue, _vaultValue);
}
```

MEV Attack Vector:
➔ deposit()/updateVaultValue()/withdraw()

# VN004-2: Potential Sandwich/MEV Attacks in Vault Contracts

- Affected: UnifiedSeniorVault::rebase()

```
function rebase(uint256 lpPrice) public virtual onlyAdmin {
    ...
    // Step 5: Update rebase index
    uint256 oldIndex =  rebaseIndex;
    _rebaseIndex = FeeLib.calculateNewRebaseIndex(oldIndex, selection.selectedRate)
    _epoch++;

    MEV Attack Vector:
     ➔   deposit()/rebase()/withdraw()

                                                (management + performance)
                                        election.feeTokens;
    if (totalFeeTokens > 0) {
        _mint(_treasury, totalFeeTokens);
    }

    _lastRebaseTime = block.timestamp;

    emit Rebase(_epoch, oldIndex, _rebaseIndex, totalSupply());
    emit RebaseExecuted(_epoch, selection.apyTier, oldIndex, _rebaseIndex, selectio
    emit FeesCollected(mgmtFeeTokens, selection.feeTokens);  // Now shows both fees
```

# VN005: Potential Risks Associated with Centralization

- Owner is privileged to guard/coordinate protocol-wide operations
  - Admin: Manage roles, etc
  - LiquidityManager: Manage funds and LPs
  - PriceFeedManager: Updates vault value, impacting share price and returns
  - ContractUpdater: Updates important contract addresses for upgrades.
  - …

```
function investInLP(address lp, uint256 amount) external onlyLiquidityManager
    if (lp == address(0)) revert AdminControlled.ZeroAddress();
    if (amount == 0) revert InvalidAmount();
    if (!_isWhitelistedLP[lp]) revert WhitelistedLPNotFound();

    // Check vault has sufficient stablecoin balance
    uint256 vaultBalance = _stablecoin.balanceOf(address(this));
    if (vaultBalance < amount) revert InsufficientBalance();

    // Transfer stablecoins from vault to LP
    _stablecoin.transfer(lp, amount);
```

```
function addWhitelistedLPToken(address lpToken) external onlyAdmin {
    if (lpToken == address(0)) revert AdminControlled.ZeroAddress();
    if (_isWhitelistedLPToken[lpToken]) revert LPAlreadyWhitelisted();

    _whitelistedLPTokens.push(lpToken);
    _isWhitelistedLPToken[lpToken] = true;

    emit WhitelistedLPTokenAdded(lpToken);
}
```

# N1-1: Proper Use of previewDeposit() in BaseVault::seedVault()

- Affected: BaseVault::seedVault(), ReserveVault::seedReserveWithToken()

```
function seedVault(…
) external onlySeeder {
    ...
    // Step 1: Transfer LP tokens from seedProvider to vault
    IERC20(lpToken).safeTransferFrom(seedProvider, address(this), amount);
    // Step 2: Transfer LP tokens from vault to hook
    IERC20(lpToken).safeTransfer(address(kodiakHook), amount);

    // Step 3: Calculate value = amount * lpPrice / 1e18
    // lpPrice is in 18 decimals, representing how much stablecoin per LP token
    uint256 valueAdded = (amount * lpPrice) / 1e18;

    // Step 4: Mint shares to seedProvider
    // Get current share price (how many assets per share)
    uint256 sharePrice = totalSupply() > 0 ? (totalAssets() * 1e18) / totalSupply() : 1e18;
```

Step 1: sharesToMint = previewDeposit(valueAdded)
Step 2: safeTransferFrom(...)
Step 3: _mint(...)
Step 4: _vaultValue += valueAdded

```
        // Calculate shares based on current share price
        sharesToMint = (valueAdded * 1e18) / sharePrice;
    }

    // Mint shares directly (bypass normal deposit flow)
    _mint(seedProvider, sharesToMint);

    // Step 5: Update vault value to include new LP value
    _vaultValue += valueAdded;
    _lastUpdateTime = block.timestamp;
```

# N1-2: Proper Use of previewDeposit() in JuniorVault::approveLPDeposit()

```
function approveLPDeposit(uint256 depositId, uint256 lpPrice) external onlyLiquidityManager {
    ...

    // Calculate value and shares
    uint256 valueAdded = (deposit.amount * lpPrice) / 1e18;

    // Calculate shares to mint (same logic as seedVault)
    uint25                                                                    ;
    uint25
                Step 1: sharesToMint = previewDeposit(valueAdded)
                Step 2: _mint(...)
                Step 3: _vaultValue += valueAdded
    if (sh
        sh
    } else {
        sharesToMint = (valueAdded * 1e18) / sharePrice;
    }

    // Mint shares to depositor
    _mint(deposit.depositor, sharesToMint);

    // Update vault value
    _vaultValue += valueAdded;
    _lastUpdateTime = block.timestamp;
```

# N2: Revisited Permission Design of BaseVault::seedVault()

- Affected: BaseVault::seedVault(), ReserveVault::seedReserveWithToken(), UnifiedSeniorVault::seedVault()

```
function seedVault(
    address lpToken,
    uint256 amount,
    address seedProvider,
    uint256 lpPrice
) external onlySeeder {
    // Validation
    if (lpToken == address(0) || seedProvider == address(0)) revert AdminControlled.Z
    if (amount == 0) revert InvalidAmount();

    // Step 1: Transfer LP tokens from seedProvider to vault
    IERC20(lpToken).safeTransferFrom(seedProvider, address(this), amount);

    // Step 2: Transfer LP tokens from vault to hook
    IERC20(lpToken).safeTransfer(address(kodiakHook), amount);
```

Excessive permissions that allow transferring assets from any authorized user are not allowed.

# N3: Improved Logic of BaseVault::removeWhitelistedLP()

```
function removeWhitelistedLP(address lp) external onlyAdmin {
    if (lp == address(0)) revert AdminControlled.ZeroAddress();
    if (!_isWhitelistedLP[lp]) revert WhitelistedLPNotFound();

    // Find and remove from array using swap-and-pop
    for (uint256 i = 0; i < _whitelistedLPs.length; i++) {
        if (_whitelistedLPs[i] == lp) {
            _whitelistedLPs[i] = _whitelistedLPs[_whitelistedLPs.length - 1];
            _whitelistedLPs.pop();
            break;
        }
    }
     Directly use _removeWhitelistedLPInternal(lp)
    // Remove from mapping
    _isWhitelistedLP[lp] = false;

    emit WhitelistedLPRemoved(lp);
```

# N4: Improved Logic of BaseVault::setMgmtFeeSchedule()

```
/**
 * @notice Set management fee schedule (time between management fee mints)
 * @dev Only admin can update the schedule
 * @param newSchedule New schedule in seconds (e.g., 7 days, 30 days)
 */
function setMgmtFeeSchedule(uint256 newSchedule) external onlyLiquidityManager {
    if (newSchedule == 0) revert InvalidSchedule();

    uint256 oldSchedule = _mgmtFeeSchedule;
    _mgmtFeeSchedule = newSchedule;
}
```

Should set a minimum interval (e.g., 30 days) to prevent mintManagementFee() from being called too frequently.

# N5: Integration of Non-Standard ERC20 Tokens

- Affected: BaseVault::investInLP()/withdrawLPTokens()/transferToSenior(), UnifiedSeniorVault::investInLP()/withdrawLPTokens()/deposit()/withdraw()/emergencyWithdraw(),

```
function investInLP(address lp, uint256 amount) external onlyLiquidityManager {
    if (lp == address(0)) revert AdminControlled.ZeroAddress();
    if (amount == 0) revert InvalidAmount();
    if (!_isWhitelistedLP[lp]) revert WhitelistedLPNotFound();

    // Check vault has sufficient stablecoin balance
    uint256 vaultBalance = _stablecoin.balanceOf(address(this));
    if (vaultBalance < amount) revert InsufficientBalance();

    // Transfer stablecoins from vault to LP
    _stablecoin.transfer(lp, amount); transfer() -> safeTranfer()

    emit LPInvestment(lp, amount);
```

# N6: Revisited Logic of UnifiedSeniorVault::previewDeposit()

```
function previewDeposit(uint256 assets) public view virtual returns (uint256) {
    return assets; // 1:1 at current index
}
  MathLib.calculateSharesFromBalance(assets, _rebaseIndex)
```

# N7: Improper Rounding Direction in UnifiedSeniorVault::_burn()

```
function _burn(address from, uint256 amount) internal virtual {
    if (from == address(0)) revert InvalidRecipient();
                                                    rounding up instead of down
    uint256 sharesToBurn = MathLib.calculateSharesFromBalance(amount, _rebaseIndex);

    if (_shares[from] < sharesToBurn) revert InvalidAmount(); // Use IVault error

    unchecked {
        _shares[from] -= sharesToBurn;
        _totalShares -= sharesToBurn;
    }

    emit Transfer(from, address(0), amount);
}
```

# N8: Suggested Slippage Protection in depositLP()

- Affected: JuniorVault/UnifiedSeniorVault::depositLP()

```
function depositLP(address lpToken, uint256 amount) external returns (uint256 depositId)
    if (lpToken == address(0)) revert AdminControlled.ZeroAddress();
    if (amount == 0) revert InvalidAmount();
    if (address(kodiakHook) == address(0)) revert KodiakHookNotSet();

    // Transfer LP from user to hook
    IERC20(lpToken).safeTransferFrom(msg.sender, address(kodiakHook), amount);

    // Create pending deposit
    depositId = _nextDepositId++;
    uint256 expiresAt = block.timestamp + DEPOSIT_EXPIRY_TIME;

    _pendingDeposits[depositId] = PendingLPDeposit({
        depositor: msg.sender,
        lpToken: lpToken,
        amount: amount,
        timestamp: block.timestamp,
        expiresAt: expiresAt,
        status: DepositStatus.PENDING
```

uint256 minShares, used in approveLPDeposit()

# N9-1: Redundant State/Code Removal

- Affected: BaseVault::setSeniorVault()/updateSeniorVault()

```
function setSeniorVault(address seniorVault_) external onlyAdmin {
    if (_seniorVault != address(0) && _seniorVault != address(0x1)) {
        revert SeniorVaultAlreadySet();
    }
    if (seniorVault_ == address(0)) revert AdminControlled.ZeroAddress();

    _seniorVault = seniorVault_;
}
```

Duplicate functions

```
function updateSeniorVault(address seniorVault_) external onlyAdmin {
    if (seniorVault_ == address(0)) revert AdminControlled.ZeroAddress();

    _seniorVault = seniorVault_;
}
```

# N9-2: Redundant State/Code Removal

- Affected: UnifiedSeniorVault::setJuniorReserve()/updateJuniorReserve()

```
function setJuniorReserve(address juniorVault_, address reserveVault_) external onlyAd
    if (address(_juniorVault) != address(0) && address(_juniorVault) != address(0x1))
        revert JuniorReserveAlreadySet();
    }
    if (juniorVault_ == address(0)) revert AdminControlled.ZeroAddress();
    if (reserveVault_ == address(0)) revert AdminControlled.ZeroAddress();

    _juniorVault = IJuniorVault(juniorVault_);
    _reserveVault = IReserveVault(reserveVault_);
}
```

Duplicate functions

```
function updateJuniorReserve(address juniorVault_, address reserveVault_) external onl
    if (juniorVault_ == address(0)) revert AdminControlled.ZeroAddress();
    if (reserveVault_ == address(0)) revert AdminControlled.ZeroAddress();

    _juniorVault = IJuniorVault(juniorVault_);
    _reserveVault = IReserveVault(reserveVault_);
}
```

# N9-3: Redundant State/Code Removal

- Affected: BaseVault/UnifiedSeniorVault::setKodiakHook()

```
function setKodiakHook(address hook) external onlyAdmin {
    // Remove old hook from whitelist if exists
    if (address(kodiakHook) != address(0)) {
        _stablecoin.forceApprove(address(kodiakHook), 0);
        Redundant, safely remove
        // Remove from whitelist
        if (_isWhitelistedLP[address(kodiakHook)]) {
            _removeWhitelistedLPInternal(address(kodiakHook));
        }
    }
```

# N9-4: Redundant State/Code Removal

- Affected: BaseVault::transferToSenior()

```
/**
 * @notice Transfer Stablecoins to Senior vault (fixes asset transfer issue)
 * @dev Called by Senior vault during backstop
 * @param amount Amount of Stablecoins to transfer
 */                          Redundant, not used anywhere
function transferToSenior(uint256 amount) external virtual onlySeniorVault {
    if (amount == 0) return;
    _stablecoin.transfer(_seniorVault, amount);
}
```

# N9-5: Redundant State/Code Removal

- Affected: ReserveVault::setKodiakRouter()/kodiakRouter()

```
/** ...
function setKodiakRouter(address router) external onlyAdmin {
    if (router == address(0)) revert ZeroAddress();
    _kodiakRouter = router;
    emit KodiakRouterSet(router);
}
```

Redundant, _kodiakRouter is not used anywhere

```
/** ...
function kodiakRouter() external view returns (address) {
    return _kodiakRouter;
}
```

# N10: Revisited Logic of sweepToKodiak()

```
function sweepToKodiak(…
) external onlyLiquidityManager {
    if (address(kodiakHook) == address(0)) revert KodiakHookNotSet();

    // Get all idle stablecoin balance
    uint256 idle = _stablecoin.balanceOf(address(this));
    if (
    ...
    // V
    uint256 lpAfter = kodiakHook.getIslandLPBalance();
    uint256 lpReceived = lpAfter - lpBefore;
    if (lpReceived < minLPTokens) revert SlippageTooHigh();

    emit KodiakDeployment(idle, lpReceived, block.timestamp);
}
```

If there is new deposit before the sweepToKodiak tx in the same block, the minLPTokens might be not accurate

# Q1: Revisited Cooldown Design in UnifiedSeniorVault

- Unclear cooldown purpose: should it update on token transfers, deposits, or withdrawals?

```
function canWithdrawWithoutPenalty(address user) public view virtual returns (bool) {
    uint256 cooldownTime = _cooldownStart[user];
    if (cooldownTime == 0) return false;
    return (block.timestamp - cooldownTime) >= MathLib.COOLDOWN_PERIOD;
}

function calculateWithdrawalPenalty(
) public view virtual returns (uint256
    uint256 cooldownTime = _cooldownSta
    if (cooldownTime == 0) {
        return FeeLib.calculateWithdrawalPenalty(amount, 0, block.timestamp);
    }
    return FeeLib.calculateWithdrawalPenalty(amount, cooldownTime, block.timestamp);
}
```

If cooldownTime is 0, will have penalty.

Actually, if cooldownTime is 0, no penalty.

# Q2: Revisited Purpose of Burn-Related Functions in BaseVault

```
/**…
function burn(uint256 amount) public virtual {
    _burn(msg.sender, amount);
}
```

What's the purpose of burn-related functions? Donation?

```
/**…
function burnFrom(address account, uint256 amount) public virtual {
    _spendAllowance(account, msg.sender, amount);
    _burn(account, amount);
}

/**…
function adminBurn(address account, uint256 amount) public virtual onlyAdmin {
    _burn(account, amount);
}
```

# Q3: Revisited Rebase Frequency in UnifiedSeniorVault

- The rebase() function assumes a monthly schedule, but _minRebaseInterval is defaulted to 30 seconds and configurable, creating a mismatch between the intended monthly rebase and the actual callable frequency.

```
uint256 internal _minRebaseInterval; // Settable by admin, default 30 seconds

function rebase(uint256 lpPrice) public virtual onlyAdmin {
    if (block.timestamp < _lastRebaseTime + _minRebaseInterval) {
        revert RebaseTooSoon();
    }

    if (lpPrice == 0) revert InvalidAmount();

    uint256 currentSupply = totalSupply();
    if (currentSupply == 0) revert InvalidAmount();

    // Step 1: Calculate management fee tokens to mint (1% annual / 12 months)
    uint256 mgmtFeeTokens = FeeLib.calculateManagementFeeTokens(_vaultValue);

    // Step 2 & 3: Dynamic APY selection (using full vault value, not reduced)
    RebaseLib.APYSelection memory selection = RebaseLib.selectDynamicAPY(
        currentSupply,
        _vaultValue  // Use full vault value now!
    );
```

# Q4: Revisited _*vaultValue* Initialization in Vault Contracts

- Under what scenarios is the initial value of _vaultValue not zero?

```
function __BaseVault_init( …
) internal onlyInitializing {
    if (stablecoin_ == address(0)) revert AdminControlled.ZeroAddress()

    __ERC20_init(vaultName_, vaultSymbol_);
    __ERC4626_init(IERC20(stablecoin_));
    __AdminControlled_init();

    _stablecoin = IERC20(stablecoin_);
    _seniorVault = seniorVault_; // Can be placeholder initially
    _vaultValue = initialValue_;
    _lastUpdateTime = block.timestamp;

    // Initialize management fee minting variables
    _lastMintTime = block.timestamp;
    _mgmtFeeSchedule = 30 days; // Default: 30 days
```

# Q5: Non-18-Decimal Tokens Would Break the Valuation Math

```
function deposit(uint256 assets, address receiver) public virtual
    ...

    // Transfer stablecoin from user
    _stablecoin.transferFrom(msg.sender, address(this), assets);

    // Track capital inflow: increase vault value by deposited as
    _vaultValue += assets;

    // Mint snrUSD to receiver (1:1 at current index)
    _mint(receiver, assets);

    emit Deposit(receiver, assets, assets);
```

```
function approveLPDeposit(uint256 depositId, uint256 lpPrice) external
    ...
    // Calculate value and shares (Senior: 1:1 with USD)
    uint256 valueAdded = (pendingDeposit.amount * lpPrice) / 1e18;
    uint256 sharesToMint = valueAdded; // Senior mints 1:1

    // Mint shares to depositor
    _mint(pendingDeposit.depositor, sharesToMint);

    // Update vault value
    _vaultValue += valueAdded;
    _lastUpdateTime = block.timestamp;
```

# LiquidRoyalty (Out-of-Scope)

# VN001: Improper New Rebase Index Calculation in calculateNewRebaseIndex()

```
function rebase(uint256 lpPrice) public virtual onlyAdmin {
    ...

    // Step 5: Update rebase index
    uint256 oldIndex =  rebaseIndex;
    _rebaseIndex = FeeLib.calculateNewRebaseIndex(oldIndex, selection.selectedRate);
    _epoch++;
```

```
function calculateNewRebaseIndex(…
) internal pure returns (uint256 newIndex) {
    // I_new = I_old × (1 + r_selected × 1.02)
    // Multiplier = 1 + (monthlyRate × .02)
    uint256 multiplier = MathLib.PRECISION +
        ((monthlyRate * (MathLib.PRECISION + MathLib.PERF_FEE)) / MathLib.PRECISION);

    return (oldIndex * multiplier) / MathLib.PRECISION;
}
```

newRebaseIndex should exclude PERF_FEE, otherwise all holders' balances are incorrectly increased.

# VN002: Improper New Rebase Supply Calculation in calculateRebaseSupply()

```
function rebase(uint256 lpPrice) public virtual onlyAdmin {
    ...

    // Step 1: Calculate management fee tokens to mint (1% annual / 12 mon
    uint256 mgmtFeeTokens = FeeLib.calculateManagementFeeTokens(_vaultValu

    // Step 2 & 3: Dynamic APY selection (using full vault value, not redu
    RebaseLib.APYSelection memory selection = RebaseLib.selectDynamicAPY(
        currentSupply,
        vaultValue  // Use full vault value now!
    );
```

```
function selectDynamicAPY(
    uint256 currentSupply,
    uint256 netVaultValue
) internal pure returns (APYSelection memory selection) {
    if (currentSupply == 0) revert InvalidSupply();
    if (netVaultValue == 0) revert InvalidVaultValue();

    // Try 13% APY first (greedy maximization)
    (uint256 newSupply13, uint256 userTokens13, uint256 feeTokens13) =
        FeeLib.calculateRebaseSupply(currentSupply, MathLib.MAX_MONTHLY_RATE);
```

```
function calculateRebaseSupply(…
) internal pure returns (…
) {
    // S_users = S × r_month
    userTokens = (currentSupply * monthlyRate) / MathLib.PRECISION;

    // S_fee = S_users × 0.02
```

newSupply should include mgmtFeeTokens

```
    // S_new = S + S_users + S_fee
    newSupply = currentSupply + userTokens + feeTokens;

    return (newSupply, userTokens, feeTokens);
```

# VN003: Lack of Slippage Control in liquidateLPForAmount()

```
function _withdraw(…
) internal virtual override {
    ...
    for (uint256 i = 0; i < maxAttempts; i++) {
        ...
        // Call hook to liquidate LP with smart estimation
        try kodiakHook.liquidateLPForAmount(needed) {…
        } catch {…
        }
    }
```

```
function liquidateLPForAmount(uint256 unstake_usd) public onlyVault {
    ...
    // Query Island pool directly for actual balances
    (, uint256 honeyInPool) = island.getUnderlyingBalances();
    uint256 totalLPSupply = island.totalSupply();

    if (totalLPSupply == 0 || honeyInPool == 0) return;

    // Ca                                    actually get back as stablecoin)
    // honeyPerLP is in 1e18 precision
    uint256 honeyPerLP = (honeyInPool * 1e18) / totalLPSupply;

    // Calculate LP needed to get unstake_usd worth of HONEY
    uint256 lpNeeded = (unstake_usd * 1e18) / honeyPerLP;
```

**Lack of slippage control**

# N1: Redundant State/Code Removal

- Affected: ConcreteJuniorVault/ConcreteReserveVault/UnifiedConcreteSeniorVault::initializeV2()
- No separate initializeV2() is needed; perform this setup directly in initialize().

```
function initializeV2(
    address liquidityManager_,
    address priceFeedManager_,
    address contractUpdater_
) external reinitializer(2) {
    _liquidityManager = liquidityManager_;
    _priceFeedManager = priceFeedManager_;
    _contractUpdater = contractUpdater_;
}
```