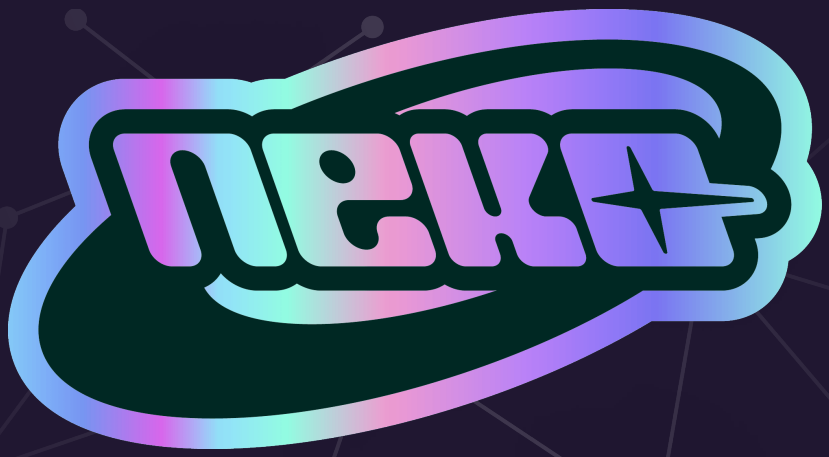
SB SECURITY

Neko.hl

Security Review



Nov 26, 2025

Contents

1. About SBSecurity

SBSecurity is a team of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

A time-boxed security review of the **omo-protocol/neko-vault** repository was conducted by **SBSecurity**. The review was performed by a team of 3 security researchers, who identified **12 issues** in total.

Overview

Project	Neko.hl
Commit Hash	6515c84b0e181c0434885a9d119ed1891f9eccce
Resolution	6c4b35853cddc37463a0a416003e76f887ab3a79
Timeline	November 18, 2025 - November 20, 2025

Scope

VaultComposerSync.sol, ShareOFT.sol, AssetOFT.sol, ShareOFTAdapter.sol

UniversalAdapterEscrow.sol

Post Audit Condition

The codebase underwent significant changes and new components were added after the initial 2 audits, yet there are still numerous issues reported.

SBSecurity does not approve this codebase as deployment ready. Additional audits are recommended before considering deployment.

5. Findings

Issues Found

Critical Risk	2
High Risk	4
Medium Risk	5
Low/Info Risk	1

ID	Title	Severity	Resolution
[C-01]	double withdrawal blocks the redemptions	Critical	Fixed
[C-02]	Proportionally strategy adjustment will brick the protocol	Critical	Fixed
[H-01]	allocations will be messed up	High	Fixed
[H-02]	When x > totalExternalDeposits deallocation will revert with underflow	High	Fixed
[H-03]	if withdrawn > externalDeposits of strategy, storage becomes incorrect	High	Fixed
[H-04]	realAssets will have sharp drops, allowing users to profit from the share price volatility	High	Fixed
[M-01]	`VaultComposerSync::quoteSend` won't work	Medium	Fixed
[M-02]	Deposits will fail when underlying strategy is unresponsive	Medium	Fixed
[M-03]	Deposits will fail when the pre-configured amount in the Calls argument isn't available as balance	Medium	Fixed
[M-04]	high unbounded gas consumption of the deallocation can prevent cross-chain calls	Medium	Fixed
[M-05]	withdrawalsExecuted isn't reset if strategy withdrawal fails	Medium	Fixed
[L-01]	syncExternalDeposits will still not sync correctly	Low	Fixed

Critical severity

[C-01] double withdrawal blocks the redemptions

Severity: Critical Risk

Description: Deallocation will try to transfer the surplus that's withdrawn from strategies twice. The problem is that the balance of the Adapter will become insufficient to cover the assets that should be pulled from the Vault as part of the original redeem process. Thus, the redeem flow is bricked forever and nobody can withdraw, unless it's from `forceDeallocate`.

```
if (caller != FORCE_DEALLOCATE_SELECTOR && withdrawalsExecuted) {
    uint256 _bal = IERC20(asset).balanceOf(address(this));
    if (_bal > assets) {
        SafeERC20Lib.safeTransfer(asset, parentVault, _bal - assets);
    }
}

// SECURITY FIX (security_issues_5nov2025_6.md Issue #1): Forward surplus to vault
// If withdrawals returned more than requested assets, forward the surplus immediately.
// This prevents externalDeposits from staying overstated and underpricing totalAssets.
// Done after slippage checks so availability checks use the full balance.
// Only forward if we actually executed withdrawals (not in Scenario 1 where balance covers all).
if (caller != FORCE_DEALLOCATE_SELECTOR && withdrawalsExecuted) {
    uint256 _bal = IERC20(asset).balanceOf(address(this));
    if (_bal > assets) {
        SafeERC20Lib.safeTransfer(asset, parentVault, _bal - assets);
    }
}
```

Recommendation: Leave only 1 surplus transfer.

Resolution: Fixed

[C-02] Proportionally strategy adjustment will brick the protocol

Severity: Critical Risk

Description: `syncExternalDeposits` tries to find the ratio between the old total deposited in external strategies vs. the new one that the admin gives as an argument, then it distributes by that ratio across all the strategies individually.

The problem is that no single strategy will be equal and increasing the deposits by a precomputed ratio is wrong and will result in corrupted `externalDeposits` mapping, which is used across the entire Adapter contract.

There are various impacts: inflating the deposited amounts, which will block withdrawals when tried, deflating that will leave funds stuck in the strategies. That will lead to loss of funds for the users and DoS of certain operations.

```
function syncExternalDeposits(uint256 newTotalExternalDeposits) external onlyOwner {
    // SECURITY FIX Issue #8: Removed pause check
    // Owner must be able to fix accounting during pause for accurate emergency operations

    // SECURITY: Can only reduce, never increase (removing ghost, not creating it)
    require(newTotalExternalDeposits <= totalExternalDeposits, "Can only reduce ghost");

    // VALIDATION: New value should make sense given valuer's current report
    uint256 balance = IERC20(asset).balanceOf(address(this));
    uint256 newMinKnown = balance + newTotalExternalDeposits;

    // SECURITY FIX (security_issues_5nov2025_5.md): Use aggregated ESCROW_TOTAL valuation
    // OLD: getTotalValue(address) → O(N) strategy enumeration → gas-unsafe for high N
    // NEW: getValue(ESCROW_TOTAL_ID) → O(1) lookup → gas-bounded
    bytes32 totalId = keccak256(abi.encodePacked("ESCROW_TOTAL", address(this)));

    (bool success, bytes memory data) = valuer.staticcall{gas: VALUER_GAS_STIPEND}(
        abi.encodeWithSignature("getValue(bytes32)", totalId)
    );

    if (success && data.length >= 32) {
        uint256 valuerValue = abi.decode(data, (uint256));

        // SAFETY CHECK: New minimum shouldn't be too far below valuer value
        // Allow up to 20% below valuer for safety margin (more conservative than 10% tolerance)
        require(valuerValue >= (newMinKnown * 8000) / 10000, "New value too low vs valuer");
    }

    uint256 oldValue = totalExternalDeposits;

    // SECURITY FIX (security_issues_5nov2025_2.md): Proportionally reduce per-strategy values
    // to maintain invariant: totalExternalDeposits == sum(externalDeposits[•])
    // This prevents asymmetric updates that cause double counting (overpricing) or
    // full-idle subtraction (underpricing) in the donation filter
    if (oldValue > 0 && newTotalExternalDeposits < oldValue) {
        // Calculate reduction ratio with 1e18 precision to avoid rounding errors
        // ratio = newTotal / oldTotal
        uint256 ratio = (newTotalExternalDeposits * 1e18) / oldValue;

        // Apply ratio to all active strategies' externalDeposits
        // TODO (security_issues_5nov2025_5.md): For large N, replace this O(N) sweep
        // with a paginated sync to avoid gas limits
```

```
bytes32[] memory activeStrategyIds = activeStrategies.values();
        for (uint256 i = 0; i < activeStrategyIds.length; i++) {
            bytes32 strategyId = activeStrategyIds[i];
            uint256 currentPerStrategy = externalDeposits[strategyId];

            if (currentPerStrategy > 0) {
                // Proportionally reduce: newValue = currentValue * ratio
                uint256 newPerStrategy = (currentPerStrategy * ratio) / 1e18;
                externalDeposits[strategyId] = newPerStrategy;

                // If both allocation and externalDeposits are now zero, remove from active set
                if (allocations[strategyId] == 0 && newPerStrategy == 0) {
                    _removeFromActiveStrategies(strategyId);
                }
            }
        }
    }

    totalExternalDeposits = newTotalExternalDeposits;

    // SECURITY FIX (security_issues_5nov2025_4.md Issue #3): Invalidate stale cache and refresh
    // syncExternalDeposits changes donation filter parameters, so cached valuation becomes stale
    cachedValuationTimestamp = 0;
    _updateCachedValuation();

    emit ExternalDepositsSynced(msg.sender, oldValue, newTotalExternalDeposits);
}
```

Let's say old:new total external deposits ratio is 1.2, this will increase the values of all the existing strategies by 1.2, but some of them might already be in negative valuation with funds seized/ liquidated, while others will surpass that 1.2 increase.

Recommendation: Introduce better functionality to handle the value updates.

Resolution: Fixed

High severity

[H-01] allocations will be messed up

Severity: High Risk

Description: Valuer is offchain, so it won't be updated when `_updateCachedValuation` is invoked. Therefore, the cached valuation will be wrong and will lead to a wrong share price if the `realAssets` rely on it.

`totalValue` on L1266 will be wrong, which will make the calculations and, most importantly, `cachedValuation` after that inaccurate as well.

```
function _updateCachedValuation() internal {
    uint256 balance = IERC20(asset).balanceOf(address(this));

    // Donation-resistant calculation (same as realAssets)
    uint256 allocatedInAdapter = totalAllocations > totalExternalDeposits
        ? totalAllocations - totalExternalDeposits
        : 0;

    uint256 excessIdle = balance > allocatedInAdapter
        ? balance - allocatedInAdapter
        : 0;

    // SECURITY FIX (security_issues_5nov2025_3.md Issue #1): Use single aggregated strategy ID
    bytes32 totalId = keccak256(abi.encodePacked("ESCROW_TOTAL", address(this)));

    // Try to get fresh valuation
    (bool success, bytes memory data) = valuer.staticcall{gas: VALUER_GAS_STIPEND}(
        abi.encodeWithSignature("getValue(bytes32)", totalId)
    );

    if (success && data.length >= 32) {
        uint256 totalValue = abi.decode(data, (uint256));

        // SECURITY FIX (security_issues_5nov2025_6.md Issue #2): Semantic-agnostic adjustment
        // Same logic as realAssets() to handle both valuer semantic interpretations
        uint256 totalValueAdj;
        if (totalValue >= excessIdle) {
            totalValueAdj = totalValue - excessIdle;
        } else {
            totalValueAdj = totalValue + allocatedInAdapter;
        }

        if (totalValueAdj > 0) {
            // Update cache with fresh valuation
            cachedValuation = totalValueAdj;
            cachedValuationTimestamp = block.timestamp;
        }
    }
    // If valuer call fails, keep existing cache (don't update)
}
```

This happens because valuer can't be updated in between smart contract calls and will be stale in each of the functions it's invoked:

- allocate

- deallocate

- executeStrategy
- executeStrategyWithSlippage
- executeStrategyBypassCircuitBreaker
- executePreConfigured

This can be exploited, since `realAssets` if return the `cachedValuation` will report wrong, thus share price will also be wrong, allowing malicious users to exploit the difference, by entering at a low price, exiting high.

Recommendation: Introduce new logic to handle the inconsistencies.

Resolution: Fixed

[H-02] When x > totalExternalDeposits deallocation will revert with underflow

Severity: High Risk

Description: When `x > totalExternalDeposits`, it will be bigger than `d` (externalDeposits of strategy) every time, then in the if case (x > 0) `externalDeposits[strategyId] = d - x == externalDeposits[strategyId] - totalExternalDeposits` will be underflowing.

This will block all the deallocation calls that withdraw from the underlying strategies and principal + yield is being withdrawn. The most realistic scenario is when there's 1 strategy, which has accumulated yield. When withdrawal happens, which touches the yield, the above-mentioned precondition is true and transaction reverts, leaving the yield permanently stuck in the strategy.

```
if (balanceAfter > adapterBalance) {
    uint256 d = externalDeposits[strategyId];
    uint256 x = balanceAfter - adapterBalance;

    // Cap reduction to requested assets (don't over-reduce if we got extra)
    if (x > assets) x = assets;

    // Cap reduction to current per-strategy external deposits (prevent underflow)
    if (x > d) x = d;//EXPLAIN: x - d is left here

    // Cap reduction to totalExternalDeposits (prevent underflow from desync)
    if (x > totalExternalDeposits) x = totalExternalDeposits;

    // Apply symmetric reduction
    if (x > 0) {
        externalDeposits[strategyId] = d - x;
        totalExternalDeposits -= x;
    }
}
```

Recommendation: Add logic to prevent the underflow scenarios in both `deallocate` and `executeStrategyWithSlippage`.

Resolution: Fixed

[H-03] if withdrawn > externalDeposits of strategy, storage becomes incorrect

Severity: High Risk

Description: When a deallocation call happens and withdrawn touches the yield, the externalDeposits mapping becomes incorrect.

```
if (balanceAfter > adapterBalance) {
    uint256 d = externalDeposits[strategyId];
    uint256 x = balanceAfter - adapterBalance;

    // Cap reduction to requested assets (don't over-reduce if we got extra)
    if (x > assets) x = assets;

    // Cap reduction to current per-strategy external deposits (prevent underflow)
    if (x > d) x = d;//EXPLAIN: x - d is left here

    // Cap reduction to totalExternalDeposits (prevent underflow from desync)
    if (x > totalExternalDeposits) x = totalExternalDeposits;

    // Apply symmetric reduction
    if (x > 0) {
        externalDeposits[strategyId] = d - x;
        totalExternalDeposits -= x;
    }
}
```

The problem is that it won't be decreased at all, because `d == x`. As a result, yield is not accounted and `externalDeposits` will contain only ghost funds that can't be touched, because they're simply missing and were withdrawn.

Recommendation: Introduce better logic to handle the yield withdrawal and distribution.

Resolution: Fixed

[H-04] realAssets will have sharp drops, allowing users to profit from the share price volatility

Severity: High Risk

Description: `realAssets` can return either `totalValueAdj`, which is the real reported value, or `cachedValuation`. The problem is that both of the values will be different, and the cached one is stale, even a second lag behind is considered stale.

This opens an attack vector for malicious users to profit from the sharp share price change, by either depositing and withdrawing or arbitrage it, by buying OTC. If they arbitrage it, the gap won't be closed until the valuer is updated and the reporting is correct again, which is a big window for profiting from the vault.

```
if (totalValueAdj > 0) {
        return totalValueAdj;
    }

    // Valuer returned 0 - check if this is legitimate (no allocations) or error
    // If no allocations, 0 is the correct value (cold start or fully deallocated)
    if (totalAllocations == 0) {
        return 0;  // Legitimate 0 value when nothing allocated
    }

    // Valuer returned 0 but we have allocations - attempt time-bounded fallback
    // If cached valuation is recent (< 4 hours), use it as fallback
    // Otherwise revert to prevent stale pricing
    if (cachedValuationTimestamp > 0 && block.timestamp - cachedValuationTimestamp <=
MAX_CACHED_VALUATION_AGE) {
        // Recent cache available - use it to maintain availability
        return cachedValuation;
    }

    // No recent cache and valuer returned 0 despite allocations - must revert
    revert ValuationUnavailable();
}
```

Recommendation: Avoid using cached values for vault-critical functions.

Resolution: Fixed

Medium severity

[M-01] `VaultComposerSync::quoteSend` won't work

Severity: Medium Risk

Description: `quoteSend` won't be working, and users won't be able to see the messaging values needed for a successful cross-chain call.

```
function quoteSend(
    address _from,
    address _targetOFT,
    uint256 _vaultInAmount,
    SendParam memory _sendParam
) external view virtual returns (MessagingFee memory) {//TODO: check if previews return correct
    /// @dev When quoting the asset OFT, if the input is shares, SendParam.amountLD must be assets (and
vice versa)

    if (_targetOFT == ASSET_OFT) {
        uint256 maxRedeem = VAULT.maxRedeem(_from);
        if (_vaultInAmount > maxRedeem) {
            revert ERC4626.ERC4626ExceededMaxRedeem(_from, _vaultInAmount, maxRedeem);
        }

        _sendParam.amountLD = VAULT.previewRedeem(_vaultInAmount);
    } else {
        uint256 maxDeposit = VAULT.maxDeposit(_from);
        if (_vaultInAmount > maxDeposit) {
            revert ERC4626.ERC4626ExceededMaxDeposit(_from, _vaultInAmount, maxDeposit);
        }

        _sendParam.amountLD = VAULT.previewDeposit(_vaultInAmount);
    }
    return IOFT(_targetOFT).quoteSend(_sendParam, false);
}
```

This happens because per `EIP4626`, `maxDeposit`/`maxRedeem` functions should return `uint256.max`, if there's no cap, which is the case here, instead they return 0 here:

```
/// @dev Gross underestimation because being revert-free cannot be guaranteed when calling the gate.
function maxDeposit(address) external pure returns (uint256) {
    return 0;
}

/// @dev Gross underestimation because being revert-free cannot be guaranteed when calling the gate.
function maxMint(address) external pure returns (uint256) {
    return 0;
}

/// @dev Gross underestimation because being revert-free cannot be guaranteed when calling the gate.
function maxWithdraw(address) external pure returns (uint256) {
    return 0;
}

/// @dev Gross underestimation because being revert-free cannot be guaranteed when calling the gate.
function maxRedeem(address) external pure returns (uint256) {
    return 0;
}
```

Recommendation: (a) Override the max functions to return non-zero amounts - not recommended, or (b) override the quote functions to omit max checks.

Resolution: Fixed

[M-02] Deposits will fail when underlying strategy is unresponsive

Severity: Medium Risk

Description: If either one of the underlying strategies isn't responsive - paused, hit supply cap, no more deposits will be accepted because the allocation multicall will revert directly.

```
function _executeMulticall(bytes32 strategyId, Call[] memory calls, bool bypassCircuitBreaker) internal {
    // L-16 Fix: Removed daily limit tracking logic per recommendation
    // Daily limits were problematic and could prevent emergency operations

    // CRITICAL FIX: Track balance before execution to detect token movements
    uint256 balanceBefore = IERC20(asset).balanceOf(address(this));

    for (uint256 i = 0; i < calls.length; i++) {
        Call memory call = calls[i];
...MORE CODE
        // Execute the call
        (bool success, bytes memory returnData) = call.target.call{value: call.value}(call.data);
        if (!success) {
            // Best-effort during deallocate: when withdrawTarget is set, skip failed subcalls
            // to allow subsequent calls to recover sufficient assets. For other flows,
            // preserve strict revert-on-failure semantics.
            if (withdrawTarget == 0) revert CallFailed(i, returnData);
        }
```

Note that this occurs only when a `liquidityAdapter` is configured.

Recommendation: To reduce the complexity to a minimum, remove all the external calls in the allocate function, keep the assets in the vault until manually allocated by agents

Resolution: Fixed

[M-03] Deposits will fail when the pre-configured amount in the calls argument isn't available as balance

Severity: Medium Risk

Description: If the liquidityData of VaultV2, where there's an encoded struct, containing in the Calls array, has an amount that is not available as a balance, strategy deposits will fail due to insufficient balance, which will disable vault deposits.

```
function enter(uint256 assets, uint256 shares, address onBehalf) internal {
    require(canReceiveShares(onBehalf), ErrorsLib.CannotReceiveShares());
    require(canSendAssets(msg.sender), ErrorsLib.CannotSendAssets());

    SafeERC20Lib.safeTransferFrom(asset, msg.sender, address(this), assets);
    createShares(onBehalf, shares);
    _totalAssets += assets.toUint128();
    emit EventsLib.Deposit(msg.sender, onBehalf, assets, shares);

    if (liquidityAdapter != address(0)) allocateInternal(liquidityAdapter, liquidityData, assets);
}
```

```
function allocate(
    bytes memory data,
    uint256 assets,
    bytes4,
    address
) external override onlyVault notPaused returns (bytes32[] memory ids, int256 change) {
    if (data.length == 0) revert InvalidData();

    // Decode allocation data
    (bytes32 strategyId, , bool executeNow, Call[] memory calls) =
        abi.decode(data, (bytes32, uint256, bool, Call[]));

    // Validate strategy exists and is active
    if (!strategies[strategyId].active) revert StrategyNotActive();

    // L-13 FIX: Use full assets amount to prevent locked tokens
    // Unlike old architecture where only partial amount was used, we utilize 100% of transferred assets
    // This prevents the issue where assets > amount would leave tokens stuck in adapter
    if (assets == 0) revert InvalidAmount();

    // SIMPLIFIED ALLOCATION: Standard ERC20 tokens only
    // Fee-on-transfer and rebase tokens are not supported by underlying protocols
    // (Morpho Vault, Pendle, etc.) so we don't need complex tracking logic

    // Update allocation tracking with transferred amount
    allocations[strategyId] += assets;
    totalAllocations += assets;

    // Add to active strategies if not already present (O(1) operation)
    activeStrategies.add(strategyId);

    // Optionally execute strategy immediately after allocation
    if (executeNow && calls.length > 0) {
        _executeMulticall(strategyId, calls, false);
    }
```

If assets < cumulative calls.value from all the deposits that are made, all the vault enter functions will be reverted.

Note that this occurs only when a `liquidityAdapter` is configured.

Recommendation: To reduce the complexity to a minimum, remove all the external calls in the allocate function, keep the assets in the vault until manually allocated by agents

Resolution: Fixed

[M-04] high unbounded gas consumption of the deallocation can prevent cross-chain calls

Severity: Medium Risk

Description: `deallocate` call has unbounded gas consumption, multicalls for strategy withdrawals on top of the existing gas-intensive storage manipulations. This can hit the limit of LayerZero's cross-chain gas forwarded and prevent the vault from processing exits on the Spoke chains.

1. call struct decoding

2. N external calls for the underlying strategies

3. balance checks

4. storage variables modification (`externalDeposits`, `totalExternalDeposits`, `cachedValuation`, `allocations` and `totalAllocations`)

5. Strategy removal from mapping and array

6. 2 token transfers + N token transfers from the underlying strategies

Recommendation: Expose batched withdrawals that are executed on a time period by agents and remove all the strategy deallocation logic from that function. This will reduce both complexity and gas consumption. The negative is when there's a `liquidityAdapter, it won't be able to pull assets.

Resolution: Fixed

[M-05] withdrawalsExecuted isn't reset if strategy withdrawal fails

Severity: Medium Risk

Description: If `externalExecuteMulticall` fails, the catch statement will be entered, continuing the execution of `deallocate`. The problem is that `withdrawalsExecuted` isn't reset to `false` and it is assumed that funds are withdrawn from the strategies

```
} else {
  // Scenario 2: Insufficient balance - need to withdraw from protocol
  // SECURITY FIX Issues #1 & #2: Removed restrictive valuer cap

  // SECURITY FIX Issue #7: Try-catch multicall to prevent revert-on-failure DoS
  // If protocol withdrawal fails (no liquidity, paused, etc.), we can still
  // return whatever balance we have instead of reverting entire deallocate
  if (withdrawCalls.length > 0) {
      withdrawalsExecuted = true; // Mark that we executed withdrawals

      // SECURITY FIX Issue #1 (security_issues_5nov2025.md): Set early-exit target
      // Enables _executeMulticall to break early when sufficient assets recovered
      withdrawTarget = assets;
      try this.externalExecuteMulticall(strategyId, withdrawCalls) {
          // Success - balance increased from protocol withdrawal
      } catch {
          // Failure - protocol couldn't provide liquidity
          // Continue with current balance (partial fulfillment)
          // Vault's transferFrom will naturally limit to available balance
      }

      // SECURITY FIX Issue #1: Reset target regardless of success/failure
      withdrawTarget = 0;
  }
```

This is problematic as further below it will either fail in the `minAmountOut` check since `deltaIncrease` will be 0 or when transfer happens and asset reverts on 0 value transfers.

```
if (
    caller != FORCE_DEALLOCATE_SELECTOR &&
    minAmountOut > 0 &&
    withdrawalsExecuted
) {
    // Measure delta increase from before withdrawCalls (adapterBalance) to current balance
    uint256 balanceAfterCheck = IERC20(asset).balanceOf(address(this));
    uint256 deltaIncrease = balanceAfterCheck > adapterBalance
        ? balanceAfterCheck - adapterBalance
        : 0;

    // Enforce minimum delta increase to protect against MEV/slippage
    // This ensures the withdrawCalls achieved acceptable execution price
    if (deltaIncrease < minAmountOut) {
        revert SlippageTooHigh();
    }
}

// SECURITY FIX (security_issues_5nov2025_6.md Issue #1): Forward surplus to vault
// If withdrawals returned more than requested assets, forward the surplus immediately.
// This prevents externalDeposits from staying overstated and underpricing totalAssets.
// Done after slippage checks so availability checks use the full balance.
// Only forward if we actually executed withdrawals (not in Scenario 1 where balance covers all).
```

```
if (caller != FORCE_DEALLOCATE_SELECTOR && withdrawalsExecuted) {
    uint256 _bal = IERC20(asset).balanceOf(address(this));
    if (_bal > assets) {
        SafeERC20Lib.safeTransfer(asset, parentVault, _bal - assets);
    }
}
```

Recommendation: Reset the `withdrawalsExecuted` to false in the catch statement if `externalExecuteMulticall` fails.

Resolution: Fixed

Low severity

[L-01] syncExternalDeposits will still not sync correctly

Severity: Low Risk

Description: If the paused state has been for a longer time, the hardcoded safety margin of 20% will prevent the owner from passing accurate values.

```
function syncExternalDeposits(uint256 newTotalExternalDeposits) external onlyOwner {
  // SECURITY FIX Issue #8: Removed pause check
  // Owner must be able to fix accounting during pause for accurate emergency operations

  // SECURITY: Can only reduce, never increase (removing ghost, not creating it)
  require(newTotalExternalDeposits <= totalExternalDeposits, "Can only reduce ghost");

  // VALIDATION: New value should make sense given valuer's current report
  uint256 balance = IERC20(asset).balanceOf(address(this));
  uint256 newMinKnown = balance + newTotalExternalDeposits;

  // SECURITY FIX (security_issues_5nov2025_5.md): Use aggregated ESCROW_TOTAL valuation
  // OLD: getTotalValue(address) → O(N) strategy enumeration → gas-unsafe for high N
  // NEW: getValue(ESCROW_TOTAL_ID) → O(1) lookup → gas-bounded
  bytes32 totalId = keccak256(abi.encodePacked("ESCROW_TOTAL", address(this)));

  (bool success, bytes memory data) = valuer.staticcall{gas: VALUER_GAS_STIPEND}(
      abi.encodeWithSignature("getValue(bytes32)", totalId)
  );

  if (success && data.length >= 32) {
      uint256 valuerValue = abi.decode(data, (uint256));

      // SAFETY CHECK: New minimum shouldn't be too far below valuer value
      // Allow up to 20% below valuer for safety margin (more conservative than 10% tolerance)
      require(valuerValue >= (newMinKnown * 8000) / 10000, "New value too low vs valuer");
  }
```

Any valuer below that or 80% of newMinKnown greater than valuer will prevent syncs. Any longer pausing period will cause the values of the strategies to eventually deviate from that if there's a lot of volatility

Recommendation: Argument is owner-controlled, so remove the checks, which would prevent the vault from resetting to the correct state after being paused.

Resolution: Fixed