BAIL security

SECURITY AUDIT

Symbiotic

Async/Account , Scope 4

FINAL REPORT

July 2026

BAILSEC.IO

Disclaimer:

BailSec was engaged by the client to perform a time-boxed technical security assessment of the target identified in this report. The assessment applies only to the scope, version, commit, deployment, materials and information identified in the report. Its findings are point-in-time and not exhaustive. Changes made after the assessment—including remediation changes—were not assessed unless expressly stated otherwise and may affect the applicability of the report's conclusions.

No security assessment can identify every vulnerability or guarantee that a system is secure, error-free or protected against attack or loss. Only the components, assumptions, threat models and risk categories expressly identified as in scope were assessed. The report should be read in full, including its scope, exclusions, unresolved findings and remediation status. The client remains responsible for design, testing, remediation, deployment, operation, monitoring and incident response.

This report is a technical assessment, not a certification, attestation, investment recommendation, or legal, financial, regulatory or tax advice. It does not constitute an endorsement of the client, the assessed system or any related project.

This report was prepared for the client under a separate engagement agreement. If published, its public availability is for transparency and informational purposes only. To the fullest extent permitted by applicable law, publication does not create an auditor-client relationship, duty of care or right of reliance in favor of any third party. Other readers should conduct their own diligence and obtain appropriate professional advice.

Only the complete, unmodified report as issued by BailSec is authoritative. The signed engagement agreement exclusively governs the contractual rights and obligations between BailSec and the client, including warranties, remedies, liability limitations, indemnities and dispute resolution. Nothing in this notice excludes liability that cannot lawfully be excluded.

1. Project Details

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

Project	Symbiotic – Async/Account – Audit Report – Scope 4
Website	symbiotic.fi
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/symbioticfi/core-mirror/blob/bab77c38dd883049723199e396d8b1d9ffd70f07/src/contracts/adapters/ll-adapter/FigureAccount.sol https://github.com/symbioticfi/core-mirror/blob/bab77c38dd883049723199e396d8b1d9ffd70f07/src/contracts/adapters/ll-adapter/OpenEdenAccount.sol
Resolution 1	

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High						
Medium	2			2		
Low	9			9		
Informational	5			5		
Governance						
Total	16			16		

2.1 Detection Definitions

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

3. Detection

FigureAccount

The FigureAccount contract is a cooldown account that redeems the Figure/Hastra token-to-redeem into the underlying redemption token through the wYLDS async redeem vault. It extends CooldownAccount and plugs the Figure-specific redemption flow into the shared cooldown lifecycle. _requestRedeem unwraps any held token-to-redeem into wYLDS, transfers the wYLDS to a freshly cloned FigureSubAccount, and initializes it so the subaccount files the redemption request. _finalizeRequests sweeps redemption tokens from subaccounts whose requests have been fully processed off-chain and removes them from the subAccounts array. _totalAssets values held wYLDS, pending redemption requests, and redemption token balances across the account and all subaccounts, converting to vault assets via the oracle when the redemption token differs from the vault asset.

Invariants

- Every entry in subAccounts has a pending redemption or an unswept redemption token balance.
- A subaccount is removed only after its pending shares reach zero, and its redemption tokens are swept.

Issue_01	wYLDS tokens stranded in subaccounts on cancellation
Severity	**Medium**
Description	The FigureAccount contract creates subaccounts for each redemption, sends the wYLDS tokens to the subaccount, and then triggers a redemption through the subaccount. If the redemption goes through, the account then draws back the redeemed tokens. However, if the redemption is cancelled, the wYLDS tokens are then stuck in the subaccount and not valued in totalAssets. There is no viable path to recover those tokens as well, leading to a material loss
Recommendations	Consider adding a path to handle redemption cancellations, allowing the FigureAccount to recover wYLDS tokens from the subaccounts.
Comments / Resolution	Acknowledged.

Issue_02	Unclaimed wYLDS Merkle rewards in FigureAccount
Severity	**Low**
Description	Figure vault shares (wYLDS) redeem 1:1 with the underlying asset. Yield is paid separately: the Figure admin deposits rewards and publishes Merkle epochs. Eligible holders must call claimRewards with valid proof to receive newly minted wYLDS. FigureAccount holds wYLDS in two places. After instant-redeeming PRIME/AUTO, it keeps wYLDS on the main account until _requestRedeem runs. It then moves wYLDS into FigureSubAccount clones while async redemption is pending. During both periods, the account can hold wYLDS for a meaningful time. The account never calls claimRewards. Claims are tied to msg.sender, so no external party can claim on its behalf. _totalAssets only counts existing wYLDS balance, not unclaimed rewards, so missed claims are invisible to Symbiotic accounting. If Figure includes FigureAccount or its subaccounts in a reward snapshot, those rewards are never collected. Symbiotic vault depositors lose yield that should have increased account assets before final redemption to USDC.
Recommendations	Add a permissionless function on FigureAccount that accepts epoch index, amount, and Merkle proof and calls claimRewards on the wYLDS vault. Invoke it from _sync or allow keepers to call it while wYLDS is held. If subaccounts can appear in reward snapshots, add the same path there.
Comments / Resolution	Acknowledged.

Issue_03	Instant redemption swap reverts when Figure requestRedeem fails
Severity	**Low**
Description	LiquidLaneAdapter._swap transfers PRIME to the redeem account and calls sync. That call is not isolated from Figure/Hastra redemption submission. When cooldown allows it, FigureAccount._requestRedeem runs two external calls in sequence: ERC4626 redeem (PRIME → wYLDS), then FigureSubAccount.initialize, which calls requestRedeem. Neither has pre-checks or error handling. Any revert propagates through sync and aborts the entire swap. Figure can revert when the vault is paused, shares are zero, a pending redemption already exists, transfers are restricted, or any other operational gate applies. If the ERC4626 redeem succeeds but initialization fails, the transaction rolls back atomically — no wYLDS is stranded and no funds are lost, but the swap still fails until Figure accepts the request.
Recommendations	Consider decoupling request submission from the swap path. Use try/catch or skip submission when Figure would reject the call. Leave idle PRIME or wYLDS in the account until a later sync can queue it.
Comments / Resolution	Acknowledged.

Issue_04	Redemption token valued at 1:1 peg without oracle when vault asset differs
Severity	Low
Description	When the redemption token is not the Symbiotic vault's underlying asset, _redemptionTokenToAssets converts amounts using decimal scaling only. It assumes one unit of the redemption token equals one unit of the vault asset, with no price oracle. For FigureAccount, wYLDS redemptions settle as USDC. If the vault also uses USDC, this conversion is skipped, and the issue does not apply. It only matters in cross-stable setups, such as a USDT vault receiving USDC redemptions. The converted value is used in totalAssets for settled balances and pending redemption amounts. That total feeds the adapter limits and share pricing. Any depeg between the two stables misstates reported assets.
Recommendations	• Prefer deployments where REDEMPTION_TOKEN matches the vault asset. • If cross-stable support is required, value redemption-token balances with a peg oracle instead of decimal scaling alone. • Document the 1:1 assumption as an explicit deployment constraint if no oracle is added.
Comments / Resolution	Acknowledged.

Issue_05	Figure redemption subaccount list can grow without a processing bound
Severity	Low
Description	FigureAccount creates a fresh cloned subaccount whenever it has wYLDS to redeem. The new clone receives the wYLDS, submits the Figure redemption request, and is then pushed into the subAccounts array. There is no maximum length for this array. There is also no cursor or bounded processing window. Both totalAssets and _finalizeRequests scan the full list every time they run. As the number of live subaccounts grows, account valuation and sync become more expensive. In normal conditions, completed requests are removed from the array. The risk appears when requests build up faster than Figure settles them, or when many small requests are created over time. The one-day cooldown limits ordinary public growth, so this is not an instant spam issue. It is more of a backlog and liveness issue. If the array becomes large enough, routine accounting and finalization can become hard/impossible to execute within practical gas limits.
Recommendations	Consider acknowledging the issue or adding bounded redemption.
Comments / Resolution	Acknowledged.

Issue_06	One failing Figure subaccount can block the whole finalization
Severity	**Informational**
Description	FigureAccount finalizes completed Figure redemptions by walking the full subAccounts array. For each tracked subaccount, it calls pendingRedemptions. If the request is complete, it reads the redemption token balance and pulls that balance back to the parent account with safeTransferFrom. This works when every subaccount behaves normally. The problem is that one failing subaccount blocks the entire loop. If pendingRedemptions, balanceOf, or safeTransferFrom reverts for a single tracked subaccount, _finalizeRequests reverts before it can process the rest of the array. This matters because sync always finalizes existing requests before it submits a new one. A single bad completed subaccount can therefore prevent healthy completed redemptions from being swept, and it can also prevent later Figure requests from being created. Since LiquidLaneAdapter calls sync during swaps, the same condition can make new swaps for that token revert until the failing subaccount is handled. This is not about ordinary pending requests. Pending requests are skipped correctly. The issue is that there is no isolation for one subaccount that starts reverting while still tracked.
Recommendations	Consider using try-catch to avoid single points of failure.
Comments / Resolution	Acknowledged.

Issue_07	Deallocate misses settled Figure redemption tokens in sub-accounts
Severity	**Informational**
Description	When a Figure redemption completes, the redemption asset is sent to the request's sub-account, not the main FigureAccount. totalAssets includes that value because it counts redemption-token balances on sub-accounts. Those tokens are only moved to the main account when sync runs _finalizeRequests. LiquidLaneAdapter.deallocate does not call sync. It only sweeps the vault asset balance on the main account. If deallocate runs before sync, settled tokens in sub-accounts are left behind even though they are already counted in adapter accounting. This creates a gap between the reported value and what can actually be withdrawn. totalAssets and freeAssets may show proceeds as available, while deallocate returns less. Funds are not lost; anyone can call sync first. The issue mainly affects operators who expect deallocation to capture all accounted proceeds in one step.
Recommendations	Document that operators should call sync on Figure accounts before relying on deallocate or freeAssets for full proceeds. Alternatively, have deallocate finalize completed Figure redemptions before sweeping balances.
Comments / Resolution	Acknowledged.

Issue_08	Missing oracle bound check
Severity	**Low**
Description	In _totalAssets, the parent Account contract values the PRIME tokens with IOracle(ORACLE).getPrice(), which uses convertToAssets but also has max/min bound checks. However, the wYLDS tokens are valued using a raw convertToAssets call, without these bound checks. *amount += IFigureYieldVault(ASYNC_REDEEM_VAULT).convertToAssets(balance);* This creates a discrepancy in the pricing algorithm, since the 2 tokens are treated differently.
Recommendations	Consider adding bound checks when valuing wYLDS tokens as well.
Comments / Resolution	Acknowledged.

FigureOracle

The FigureOracle contract is an oracle returning the Figure/Hastra redemption price in 1e18 precision, bounded by the min and max prices enforced in the Oracle base. _getPrice converts one unit of the token-to-redeem into wYLDS via ERC4626 convertToAssets, then into the redemption token via the Figure yield vault's convertToAssets, and normalizes the result by the redemption token unit.

No issues were identified in this contract.

OpenEdenAccount

The OpenEdenAccount contract is a cooldown account that redeems OpenEden HYBOND through the HYBONDExpress queued redemption flow. It extends CooldownAccount and approves the express contract for the token-to-redeem at initialization. _requestRedeem submits the account's full HYBOND balance into the express redemption queue. _finalizeRequests is a no-op because HYBONDExpress sends the redemption asset directly to the account when the queue is processed. _totalAssets values pending redemptions, queued redemptions, and settled redemption token balances; when the queue is short, it walks the queue to use per-request net asset amounts; otherwise, it falls back to valuing the queued HYBOND amount via the oracle.

Invariants

- Queued redemptions are valued at net asset amounts (redeem amount minus fee) when the queue is walkable.

Issue_09	sync can revert on sub-minimum balances blocking finalization and swaps
Severity	**Medium**
Description	The HYBONDExpress vault implements a minimum redemption amount, below which calls to requestRedeem revert. This limit is not checked in the _requestRedeem function, which is called in the sync function, so if the account currently holds less than the minimum number of tokens, it will simply revert. This is an issue since sync is called in the swap function of the LiquidLaneAdapter, so swaps will also stop due to this minimum redemption limit.
Recommendations	Consider checking the minimum redemption limit before requesting redemptions.
Comments / Resolution	Acknowledged.

Issue_10	OpenEdenAccount cannot recover escrowed HYBOND after cancelled redemptions
Severity	Low
Description	When OpenEdenAccount queues HYBOND in Express, it is the redemption sender. If Express later cancels that request and the account is banned or de-KYC'd at that time, Express does not refund HYBOND to the account. It stores the tokens in redeemEscrowBalance[OpenEdenAccount]. To recover them, someone must call claimRedeemEscrow on Express from the account itself or from an Express operator. OpenEdenAccount has no function to do this, and its multicall cannot call Express. Symbiotic has no on-chain way to recover the escrowed HYBOND. _totalAssets also ignores escrowed HYBOND, so vault NAV drops even though the tokens still belong to the account.
Recommendations	Add a permissionless claimRedeemEscrow function on OpenEdenAccount that calls Express. Include redeemEscrowBalance(address(this)) in _totalAssets until the tokens are claimed.
Comments / Resolution	Acknowledged.

Issue_11	Instant redemption swap reverts when OpenEden requestRedeem fails
Severity	**Low**
Description	LiquidLaneAdapter._swap transfers HYBOND to the redeem account and calls sync. That call is not isolated from OpenEden request submission. When cooldown allows it, CooldownAccount._sync calls _requestRedeem, and OpenEdenAccount submits the full HYBOND balance to Express.requestRedeem with no pre-checks or error handling. Any Express revert propagates through sync and aborts the entire swap, even though the failure is unrelated to the swap itself. Express can revert when redeem is paused, KYC is invalid, balance is below redeemMinimum, or offchainShares cannot cover the full amount. The last two are worse because the account always submits 100% of its balance; it cannot queue a partial amount or skip dust below the minimum. Cooldown only delays the issue. While active, _requestRedeem is not called, so small swaps succeed, and HYBOND accumulates idle. Once the cooldown expires, the next swap tries to redeem the full balance. If that total is below redeemMinimum or exceeds offchainShares, the swap reverts even though idle HYBOND is already counted in totalAssets.
Recommendations	Consider decoupling request submission from the swap path. Use try/catch or skip requestRedeem when Express would reject it.
Comments / Resolution	Acknowledged.

Issue_12	Global redeem queue length forces oracle fallback for unrelated accounts
Severity	Low
Description	When OpenEdenAccount values final-queue redemptions in totalAssets, it first checks whether the Express contract's global redeem queue is below MAX_REDEEM_QUEUE_LENGTH (30). If it is, the account walks the full queue and sums only its own entries, using each entry's locked T+2 redeemAssetAmt minus fees. If the global queue has 30 or more entries, that walk is skipped entirely. The account's own redeemInfo balance is then added to the pending-redeem bucket and priced with the current oracle rate, even though those tokens are already in the final queue with a fixed settlement amount. The mismatch is that the branch uses the total queue size across all Express users, not how many entries belong to this account. Express only exposes a global queue (getRedeemQueueLength, getRedeemQueueInfo(index)); there is no per-account index. So one account's valuation can change because of other users' redemptions, not because of its own queue state. This mainly affects totalAssets accuracy. Deposits, withdrawals, and share pricing that rely on it can drift from true backing when the global queue is full. The vault accounting can overstate or understate an account's assets depending on how far the live oracle price sits from the locked T+2 price.
Recommendations	There is no easy fix because the Express contract does not expose a way to look up redeem info by entry ID or user address, and queue indices are positional rather than fixed per request. This risk should be clearly documented.
Comments / Resolution	Acknowledged.

Issue_13	Pending-to-final redeem transition causes discontinuous NAV
Severity	**Low**
Description	In OpenEdenAccount._totalAssets, pending and final-queue redemptions are priced differently. HYBOND still in the pending queue (pendingRedeemInfo) is always converted through _tokenToRedeemToAssets using the current oracle rate from OpenEdenOracle. After OpenEden Express runs processPendingRedeems, those tokens move into the final queue (redeemInfo) and are valued from the locked T+2 snapshot: redeemAssetAmt - feeAssetAmt from each queue entry. That means the same HYBOND balance can be marked at two different asset values across one Express state change, even though no tokens have settled into the account yet. If the oracle price has moved between request time and T+2 processing, the locked amount will differ from the live oracle estimate. When the transition happens, totalAssets can jump up or down in a single step. This only affects valuation while tokens are still inside Express. It does not steal funds directly, but any deposit, withdrawal, or share conversion that reads totalAssets around that moment can mint or burn shares against a stale or suddenly shifted NAV. The timing is not controlled by OpenEdenAccount; Express's operator decides when processPendingRedeems runs, so longer delays increase the chance of oracle drift and a larger jump.
Recommendations	Document that pending redemptions are oracle-estimated while final-queue redemptions use locked T+2 amounts, and that NAV can step-change when Express moves entries between queues. Treat totalAssets as approximate around processPendingRedeems execution, especially when the oracle price has moved since the redeem was requested.
Comments / Resolution	Acknowledged.

Issue_14	Unlimited HYBOND approval granted to upgradeable Express contract
Severity	**Informational**
Description	OpenEdenAccount grants unlimited HYBOND approval to OpenEden Express at initialization. Express needs this to pull HYBOND during requestRedeem. Express is a UUPS upgradeable contract controlled by OpenEden's UPGRADE_ROLE, not Symbiotic. The approval stays active even if Express is upgraded, so Symbiotic trusts both current Express logic and any future implementation at that address. A malicious or compromised upgrade can use the existing approval to pull all HYBOND from OpenEdenAccount at any time, without calling requestRedeem. The full account balance can be stolen.
Recommendations	Approve only the amount needed for each redemption in _requestRedeem, instead of giving unlimited approval at init. Revoke the approval after each use if possible.
Comments / Resolution	Acknowledged.

Issue_15	OpenEden HYBOND has no fallback exit path
Severity	Informational
Description	OpenEdenAccount holds HYBOND as its token-to-redeem and can only dispose of it through OpenEden Express. On sync, the account calls requestRedeem on Express with its full HYBOND balance. There is no other on-chain path to move HYBOND out. The base Account contract also blocks HYBOND in convert, reverting when HYBOND is used as input. That means CoW Swap and other converter routes cannot be used as a backup. The account has no owner rescue or emergency withdrawal function. The adapter owner can only trigger sync sooner by bypassing cooldown; that still depends on Express working. Express redemption can fail or stall for several reasons: redeem pause, failed KYC checks on the account or receiver, HYBOND ban status, or Express running out of off-chain shares. When that happens, HYBOND stays in the account. Vault accounting still counts it via the oracle price in totalAssets, but it cannot be turned into a vault asset until Express accepts the redemption and later queue processing completes. This is mostly an operational dependency risk, not a direct theft bug. But if Express stays down, the account loses KYC permanently, or Express stops working for good, HYBOND in the account has no recovery route. Vault LPs would see value on paper while real liquidity is stuck.
Recommendations	Add a fallback exit for HYBOND when Express redemption is unavailable, such as owner-controlled emergency withdrawal to a designated address, and support for Express requestDirectRedeem for when the queued path fails.
Comments / Resolution	Acknowledged.

Issue_16	Immutable redemption token desyncs from Express redeem asset updates
Severity	Informational
Description	OpenEdenAccount sets REDEMPTION_TOKEN once at deployment from Express.redeemAsset. Express can later change redeemAsset via updateRedeemAsset when all queues are empty. After that change, new redemptions settle in the new token and are sent to the account. The account still only tracks the old REDEMPTION_TOKEN in _totalAssets. Queued amounts are also valued using the old token. OpenEdenOracle has the same issue. This needs a trusted Express admin action, not an external attacker. But any account deployed before the change will miss new settled funds in totalAssets, understating NAV and mispricing shares. The new token is also not auto-converted unless handled manually.
Recommendations	Read redeemAsset from Express at runtime instead of storing it immutably, or acknowledge this issue.
Comments / Resolution	Acknowledged.

OpenEdenOracle

The OpenEdenOracle contract is an oracle returning the OpenEden HYBOND net redemption price in 1e18 precision, bounded by the min and max prices enforced in the Oracle base. _getPrice calls previewRedeem on the HYBONDExpress contract for one unit of the token-to-redeem and normalizes the returned net redemption assets (after fees) by the redemption token unit.

No issues were identified in this contract.