SECURITY AUDIT

Symbiotic

Liquid Lane Adapters

FINAL REPORT

June 2026

BAILSEC.IO

Disclaimer:

Security assessment projects are time-boxed and reliant on information provided by the client, its affiliates, or its partners. The findings in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase, and apply only to the specific commit hash, version, or deployment reviewed at the time of the audit. Any subsequent modification, redeployment, or upgrade is outside the scope of this report and may invalidate its findings.

The audit does not constitute investment, legal, financial, regulatory, or tax advice, nor an endorsement of any project or team. It is based on a limited review of materials provided and is delivered on an "as-is," "where-is," and "as-available" basis. Use of blockchain technology is subject to unknown risks and flaws. The scope of this assessment is technical and contract-level; unless explicitly stated otherwise, it does not cover economic or game-theoretic risks, MEV, oracle manipulation, governance attacks, centralization or admin-key risks, off-chain components, front-end code, or the behavior of third-party protocols, dependencies, libraries, or services interacted with by the audited code.

We make no warranties, express or implied, regarding the accuracy, reliability, completeness, or availability of the report or any associated services, and disclaim all implied warranties including merchantability, fitness for a particular purpose, and non-infringement. We assume no responsibility for any product, service, software, code, or material referenced by, linked to, or accessible through the report.

This report is prepared solely for the contract owner and may not be relied upon by any third party. The contract owner is responsible for making their own decisions based on the report and should seek additional professional advice as needed. To the maximum extent permitted by law, our total aggregate liability arising from or related to this audit shall not exceed the fees actually paid for the engagement. The contract owner agrees to indemnify and hold harmless the audit firm and its personnel from any claims, damages, expenses, or liabilities arising from use of or reliance on the report or the audited smart contract.

This disclaimer and any dispute arising from the audit shall be governed by the laws of Wyoming (USA), without regard to its conflict-of-laws principles. By engaging in this audit, the contract owner acknowledges and agrees to these terms.

1. Project Details

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

Project	Symbiotic – Liquid Lane Adapters – Audit Report
Website	symbiotic.fi
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/symbioticfi/rfq/blob/f21f049a1f3b5f75100f305c0491a1fecfa22099/src/Reactor.sol https://github.com/symbioticfi/core-mirror/tree/8e5c03316c8bf7420b71abb3abb4a78c9dd9b6bd/src/contracts
Resolution 1	https://github.com/symbioticfi/rfq/blob/041aab4487e8e71f54e4de6acb7951792b7c9e51/src/Reactor.sol https://github.com/symbioticfi/core-mirror/tree/8fbde1bccc4400fc952f1484478209144dad8f06/src/contracts

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High						
Medium	9	2	1	6		
Low	20	2		18		
Informational	9		1	8		
Governance	8			8		
Total	46	4	2	40		

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

Reactor

Reactor is the RFQ settlement boundary for LiquidLane fills. It validates the swapper-signed request and protocol-signed order, consumes the swapper nonce, transfers the requested tokenIn from the swapper directly into registered LiquidLane adapters, calls the filler/executor callback, and then enforces the signed output payments.

Orders are EIP-712 signed. The request binds tokenIn, amountIn, outputs, deadline, nonce, and protocol signer. The order binds the request, swapper signature, swapper, and filler. The concrete adapter swap payloads and executor data are supplied at fill time, but every selected adapter must be a LiquidLane adapter factory entity, every selected token must match the request tokenIn, and the sum of selected direct and discount input amounts must equal request.amountIn.

After adapter execution, ERC20 outputs are pulled from the filler/executor address, while native outputs are paid from the native balance of the Reactor. Any native balance left on Reactor after output settlement is refunded to the filler/executor.

Appendix: Settlement Roles

The order.filler is the contract calling Reactor, commonly an Executor. The Executor is responsible for invoking adapter swaps and preparing/approving output assets so Reactor can settle the request outputs. Reactor does not itself verify that the current adapter proceeds funded the current output obligations.

Privileged Functions

None.

Invariants

- A swapper request nonce can be consumed only once.
- msg.sender must equal order.filler.
- The protocol signature must validate against the full order hash, and the swapper signature must validate against the request hash.
- Every selected adapter must be registered in LIQUID_LANE_ADAPTER_FACTORY.
- Every direct or discount swap leg must consume the same token as request.tokenIn.
- The aggregate selected input amount must equal request.amountIn.
- Swapper input is transferred to adapters before executor execution, and signed outputs are settled only after executor execution returns.

- ERC20 outputs are sourced from the filler/executor; native outputs are sourced from Reactor's native balance and residual native balance is refunded to the filler/executor.

Issue_01	Rfq swap outputs can be routed away while reactor pays from shared executor balances
Severity	Governance
Description	Reactor output checks can be satisfied from executor balances already present before the fill. If the real swap proceeds are routed elsewhere, the user can still be paid from shared executor funds. This applies to the bundled example Executor and any custom executor that uses pooled balances, reusable approvals, or caller-controlled swap payloads without per-fill accounting. In those designs, executor permissions and pooled balances become part of the settlement trust boundary.
Recommendations	Avoid pooled reusable balances on executors. Validate fills with per-fill balance deltas and restrict executor call construction and direct executor access to trusted paths.
Comments / Resolution	Acknowledged.

Issue_02	Reactor fills can fail if the recipient is some other address
Severity	**Low**
Description	In Reactor._fill, at the end of the execution, the output amounts are sent to the swapper. *IERC20(order.request.outputs[i].token)* *.safeTransferFrom(msg.sender,* *order.request.outputs[i].recipient, order.request.outputs[i].amount);* This basically sends the token from the filler to the swapper. The issue is that the filler, for example, the Executor contract, does not enforce that the swap proceeds go to itself. The Executor simply calls *ILiquidLaneAdapter(swapInputs[i].adapter).swap(swapInputs[i].swap);* And a recipient address is present in the swap struct inside, which receives the output tokens. For the Reactor fill to work, this recipient must be the filler itself, and not some other address.
Recommendations	Consider enforcing in the Executor, that the swap recipient is the Executor contract itself.
Comments / Resolution	Acknowledged.

Issue_03	Rfq native refund reentrancy can steal fill surplus
Severity	**Low**
Description	Reactor._fill executes the filler executor, pays outputs, and then refunds the Reactor's remaining native token balance to msg.sender. Native output transfers use a call that forwards control to the recipient. A malicious native output recipient can reenter Reactor.fill during the output transfer. If the reentrant fill is valid and reaches the refund path, it can receive native token surplus that belonged to the outer fill. The outer fill then resumes with the surplus already drained. The issue comes from refunding the entire contract balance rather than tracking the surplus for the current fill, combined with the lack of a reentrancy guard around fill settlement. This is not specific to the example Executor, but it requires an executor flow that leaves native value in the Reactor during output settlement.
Recommendations	Add reentrancy protection around fill settlement. Track native value received and spent per fill, and refund only the surplus attributable to that fill. Also consider rejecting zero-input/no-output fills unless they are explicitly required.
Comments / Resolution	Fixed by adding reentrancy protection. Native refunding still sweeps the full Reactor balance instead of per-fill surplus.

Issue_04	protocolSignature check can be bypassed by a swapper
Severity	**Low**
Description	In Reactor._fill, the contract ensures a valid protocol signature is present. *if (!SignatureChecker.isValidSignatureNow(* *order.request.protocol, _hashTypedDataV4(_hashOrder(order)),* *protocolSignature* *)) {* *revert InvalidProtocolSignature();* *}* However, the contract never ensures that order.request.protocol is actually a protocol whitelisted address. Thus, any swapper can select any address as order.request.protocol, and create a signature to go alongside it. No protocol address will be able to validate any of the parameters, since a protocol signature is redundant.
Recommendations	Consider checking the protocol address against a whitelist of protocol addresses.
Comments / Resolution	Acknowledged.

Issue_05	Arbitrary call in executor contract can be used to steal funds
Severity	**Low**
Description	In Reactor._fill, the executor/caller is called with a callback. *IExecutor(msg.sender).execute(order, swapInputs, discountSwapInputs, executorData);* In the executor, some arbitrary calls are done. *Call[] memory calls = abi.decode(executorData, (Call[]));* *for (uint256 i; i < calls.length; ++i) {* *calls[i].target.functionCallWithValue(calls[i].data, calls[i].value);* *}* Since executorData is never checked, these calls can be to tokens, moving them out of the executor contract, or from any user who had given approval to the executor contract. Thus, it is advised to be extra careful with this arbitrary call, since approvals can be tokens can be stolen in this way.
Recommendations	Consider allowing only a whitelist of arbitrary calls, preventing calls to direct tokens.
Comments / Resolution	Acknowledged.

Issue_06	Empty outputs array allows fill with no swapper compensation
Severity	**Informational**
Description	A fill is driven by a signed Request with tokenIn, amountIn, and an outputs array. After checks pass, _fill pulls the swapper's input into adapters, runs swaps, then delivers each output from the filler to its recipient. The contract never requires outputs to be non-empty. If the array is empty, the delivery loop is skipped and the fill still succeeds. The swapper loses their full input and receives nothing. The filler pays no output tokens. An empty array is valid in the EIP-712 hash, so a signed request can include it. The swapper must sign, but a buggy frontend or bad off-chain builder could produce this by mistake. The interface defines InvalidOutput, but _fill never uses it. Example: Swapper signs 100 RWA in with outputs = []. Filler fills the order. Swapper loses 100 RWA and gets 0 back. Filler pays nothing.
Recommendations	Revert in _fill when order.request.outputs.length == 0, using InvalidOutput.
Comments / Resolution	Acknowledged.

CoWSwapConverter

CoWSwapConverter is an asynchronous CoW Protocol conversion helper. It creates presigned, partially fillable CoW sell orders for tokens held by the converter contract. The sell token remains in the converter until CoW settlement pulls it through the CoW Vault Relayer allowance, and the buy token is delivered back to the converter contract as the order receiver.

Configured converters can call convert immediately. Non-converters must first call prepareConvert, wait through the execution delay for the exact current nonce/request hash, and then call convert. Each conversion consumes the nonce for tokenIn, so prepared requests are tied to the nonce that was current when they were prepared.

convert validates token pair, sell balance, buy amount, order expiry, maximum validity duration, and delayed execution requirements. It approves the CoW Vault Relayer when needed, builds the CoW order with receiver = address(this), and marks the order UID as presigned in CoW Settlement.

Appendix: Live Orders

Presigning does not reserve the sell balance locally. While an order is live, the converter still holds the sell token and local accounting may continue to see that balance until CoW settlement pulls it. Partial fills leave the unfilled sell balance in the converter and deliver only the filled buy amount back to the converter.

Privileged Functions

- setConverters
- invalidateConvert

Invariants

- convert must not sell a token into itself.
- convert must not create zero-sell orders or orders larger than the current token balance.
- Buy amount must be nonzero and validity must be within the allowed time window.
- Non-converters can execute only prepared requests for the current token nonce after the execution delay.
- Each successful convert consumes the nonce for tokenIn.
- CoW orders must use ERC20 sell and buy balances and must send bought tokens back to the converter contract.
- Converter addresses are the only accounts that can bypass the preparation delay or invalidate presigned CoW orders.

Issue_07	Unbounded converter list can make converter checks expensive
Severity	**Governance**
Description	CoWSwapConverter.setConverters lets the owner replace the converter list with an arbitrary-length array. _isConverter then scans that array linearly in convert() and invalidateConvert(). If governance configures a very large converter list, or places active converters near the end of the list, converter checks can become unnecessarily expensive and may make conversion or invalidation calls impractical.
Recommendations	Add a maximum converter count, or replace the linear membership check with a mapping such as isConverter[address]. Alternatively, acknowledge this issue.
Comments / Resolution	Acknowledged.

Issue_08	Permissionless delayed cow conversion allows attacker-chosen limit prices
Severity	Medium
Description	CoWSwapConverter.prepareConvert is permissionless and stores a delayed conversion request. convert later accepts the prepared parameters and only requires a nonzero buy amount. For account-held intermediate tokens that are convertible but not the account asset or TOKEN_TO_REDEEM, this lets an external caller choose the CoW order limit price. An attacker can prepare a conversion with a deliberately low buy amount, wait for the execution delay, and then execute the conversion. The account approves the CoW vault relayer and pre-signs a partially fillable sell order at the weak limit price. If solvers fill the order, account value can be sold too cheaply. The delay prevents immediate execution but does not validate price. The issue comes from permissionless control over conversion terms for tokens that contribute to account accounting. Trusted converter-created orders can have the same economic shape if stale or weak terms are left active, but that path is mainly a converter-policy issue. The permissionless delayed path is more relevant because the conversion terms can be chosen by any caller. The converter also does not reserve sell balances for already presigned CoW orders. Multiple live orders can therefore target the same token balance. If a weakly priced order is presigned alongside a fairer order, CoW settlement can consume the shared balance through the weaker order first, leaving the better order unfillable. The per-token nonce limits future permissionless preparation, but it does not reserve balance for orders that have already been presigned. Authorized converters can also create overlapping live orders directly.
Recommendations	Consider requiring the owner, governance, or an authorized converter to

	approve conversion parameters that affect accounted assets. Also consider enforcing a minimum buy amount derived from an oracle and slippage policy. Furthermore, consider tracking reserved sell amounts per token while CoW orders are live, or allowing only one active presigned sell order per token.
Comments / Resolution	Acknowledged.

Issue_09	Prepared cow conversions can be warehoused while the token nonce remains unchanged
Severity	Low
Description	Prepared CoW conversions can remain executable after the initial delay without a tight local expiry enforced by the converter itself. A caller can prepare conversion parameters with a future validTo value and wait until that future window to create the CoW order. This is bounded by the per-token nonce. The prepared request is stored under the current nonce for tokenIn, and any successful conversion for the same tokenIn advances that nonce. If another conversion happens first, the prepared request becomes unreachable through the permissionless path. The risk is that, for an idle token whose nonce does not change, old prepared parameters can later be used to pre-sign a CoW order without a fresh authorization or fresh price check at execution time. The same nonce behavior is also a liveness caveat, because any successful conversion can invalidate other prepared conversions for the same tokenIn.
Recommendations	Add an explicit expiry to prepared conversions and require convert to execute only within a short window after the delay. Consider requiring fresh owner-signed or governance-approved conversion data at execution time, and enforce oracle-based minimum outputs using current prices rather than only prepared parameters. Document that nonce advancement can invalidate outstanding prepared conversions for the same tokenIn.
Comments / Resolution	Fixed by allowing converter to invalidate a nonce.

Issue_10	prepareConvert always reserves under the current nonce, preventing future order queuing
Severity	Low
Description	CoWSwapConverter::prepareConvert records a pending order at executableAt[nonces(tokenIn)][requestHash], always using the latest value of nonces(tokenIn). Whenever an order is executed, the nonce advances. Thus, orders can be queued only at the latest nonce, which advances on any execution. So multiple successive orders can never be queued, since executing one invalidates all the other queued ones.
Recommendations	Consider acknowledging this issue or allowing the caller to choose the nonce.
Comments / Resolution	Acknowledged.

LiquidLaneAdapter

LiquidLaneAdapter is the single-vault adapter for issuer-facing instant redemption flows. It manages active token-to-redeem markets, creates token accounts through AccountRegistry, prices them through account oracles, and pays vault-asset output to swap recipients.

Vault-funded liquidity comes from VaultV2 through UniversalDelegator. totalAssets sums active account value, while freeAssets sums active account vault-asset balances that can be swept on deallocation.

Swaps can be direct, signed delegated, or protocol-wrapped discount swaps. _swap validates token/rate/limit, uses acquisition prefunds first, allocates remaining vault assets if needed, routes acquired token-to-redeem to owner/market-maker receivers, sends the rest to the account, syncs it, and pays vault asset to the recipient.

Deallocation is delegator-only and sweeps realized vault-asset balances from active accounts back through the adapter so the delegator can push them into the vault.

Appendix: Acquisition Balances

Owner and market-maker acquisition balances are vault-asset prefunds used to buy a proportional share of incoming token-to-redeem. The funded output share is computed in vault-asset units, and the acquired token share is derived from tokenOutToAcquire / amountOut * amountIn.

Appendix: Limits and Discounts

limit[token] is checked against current account totalAssets plus the vault-funded output portion, tokenOutToAllocate. Swap rate validation compares requested output against oracle fair value reduced by minDiscount.

Privileged Functions

- addTokenToRedeem
- removeTokenToRedeem
- setMarketMaker
- setLimit
- setMinDiscount
- setPauser
- setUnpauser
- pause
- unpause
- setFiller

- invalidateNonce
- depositToAcquire
- withdrawToAcquire

Invariants

- Only active token-to-redeem markets can be swapped.
- totalAssets must equal the sum of active account totalAssets.
- freeAssets must equal the sum of active account vault-asset balances.
- The adapter is allocatable only during _swap while _inSwap is set.
- Requested swap output must not exceed oracle fair value after the configured minimum discount.
- Account exposure plus the vault-funded output portion must not exceed limit[token].
- Vault allocation through UniversalDelegator.allocateExact must cover the full non-acquisition output portion.
- Acquisition balances must be debited by the vault-asset amount they fund, and acquired token-to-redeem must be sent to configured receivers.
- Remaining token-to-redeem must be transferred to the account and synced before vault-asset output is paid.
- Active accounts must not be removed while their totalAssets is nonzero.
- Deallocation must return realized account vault-asset balances only through the configured delegator path.

Issue_11	Existing accounts keep stale owner and converter authority after adapter ownership transfer
Severity	Governance
Description	Transferring LiquidLaneAdapter ownership does not update already-created account owners or converter sets. The old account owner or converter can therefore retain authority over existing accounts after adapter governance has changed.
Recommendations	Make ownership transfers include an account authority migration checklist. Consider exposing a controlled flow to update active account owners and converters.
Comments / Resolution	Acknowledged.

Issue_12	Stale filler approvals reactivate when a market maker address is reused
Severity	Governance
Description	Filler approvals are keyed by market maker address. If an old market maker address is reused, old filler permissions can become active again. This is mainly relevant if fillers include more than the intended Reactor or if operational addresses are reused.
Recommendations	Clear filler approvals when rotating market makers or consider scoping filler approvals by market maker epoch.
Comments / Resolution	Acknowledged.

Issue_13	Token readd reuses stale per token state
Severity	**Governance**
Description	removeTokenToRedeem clears only part of the token's state. Account address, minDiscount, acquisition balances, nonce state, and related configuration can remain. If the token is added again, old state can affect the new activation period. If value is donated or later settles into the inactive account, readd can also create a sudden NAV increase for the then current shareholders that can be frontrun. Readd also cannot be used to rotate the token's account address. It reuses accounts[tokenToRedeem], so account replacement requires a migration path or a new adapter deployment rather than ordinary remove and add operations.
Recommendations	Reset or explicitly reinitialize per token state on readd. Otherwise document readd as a stateful continuation and require an operational review before reactivation.
Comments / Resolution	Acknowledged.

Issue_14	Donation attack via raw balanceOf accounting in LiquidLaneAdapter
Severity	**Medium**
Description	LiquidLaneAdapter and its per-token Account contracts value assets using raw balanceOf instead of tracked internal balances. Account.totalAssets() counts the vault asset and TOKEN_TO_REDEEM held at the account address, plus any strategy-specific holdings from _totalAssets(). LiquidLaneAdapter rolls this up into its own totalAssets() and freeAssets(), and the vault uses those values when pricing shares through VaultV2.getAccrueInterest() and the UniversalDelegator's adapter totals. This opens up this adapter and thus the vault to donation inflation attacks, where early depositors can inflate the share ratio of the vault. Donors can also donate tokens to the adapter directly and use them to manipulate the composition of the vault, diverting more funds to the adapter than intended and draining out the other adapters.
Recommendations	Consider switching to a stored balance account instead of using balanceOf.
Comments / Resolution	Acknowledged.

Issue_15	Discount swap has no slippage protection on execution-time oracle pricing
Severity	Medium
Description	The discount swap path in LiquidLaneAdapter prices redemptions at execution time instead of at quote time. When a swap runs, amountOut is derived from the current on-chain oracle price and the signed discount percentage. The signed payload only authorizes the discount policy, token, nonce, and deadlines. It does not fix amountIn, amountOut, or a minimum acceptable output. Because of this, a user who receives an off-chain quote can still be filled later at a worse rate, as long as both the discount deadline and the protocol deadline have not expired. If the oracle price moves against the user between quote and execution, they receive less vault asset than expected, with no on-chain check to stop it. The internal swap logic only limits how much the vault pays relative to the current oracle and minimum discount. That protects the protocol from overpaying, not the recipient from receiving less than they were quoted. By contrast, the SignedSwap path binds exact input and output amounts in the signature, so stale, unfavorable execution is blocked there. This is mainly a user-facing risk for anyone redeeming through a discount swap expecting a guaranteed quote. The market maker, filler, or executor chooses when to submit the transaction, so delayed or strategic execution can pass oracle drift to the recipient. **Example**: A user is quoted 100 token-to-redeem for 95 vault assets at a 5% discount when the oracle price is 1:1. They approve the swap and wait. Before execution, the oracle drops, so the fair rate is 0.90 vault assets per token. The swap still succeeds within the deadline and pays about 85.5 vault assets (100 × 0.90 × 0.95). The user loses roughly 9.5 vault assets compared with the original quote, with no revert.
Recommendations	Consider allowing the caller to specify a slippage amount.
Comments / Resolution	Acknowledged.

Issue_16	Swap limit check uses a pre-deallocation account snapshot, double-counting funds, and reverting valid swaps
Severity	Medium
Description	LiquidLaneAdapter::_swap enforces a per-token exposure cap before funding the vault-backed portion of a redemption *if (IAccount(accounts[swap.tokenIn]).totalAssets() + tokenOutToAllocate > limit[swap.tokenIn]) {* * revert LimitExceeded();* *}* The check reads account.totalAssets() and adds tokenOutToAllocate to bound the account's projected exposure against the limit. The intent is to stop a single token's redemption account from accumulating more than the curator-set cap. However, the value used in the check is a snapshot taken before allocateExact runs, and allocateExact can itself reduce the account's totalAssets. Say limit[tokenIn] is 1,000,000 and the account currently holds 600,000 in realized vault asset, so account.totalAssets() returns 600,000. A swap needs tokenOutToAllocate of 500,000, but the vault has no free assets, so funding it deallocates the account's 600,000 back to the vault first. After the swap the account's totalAssets would actually be near 500,000 (its prior holdings swept out, only the freshly redeemed token-in left), well under the limit. Yet the check evaluates 600,000 + 500,000 = 1,100,000 > 1,000,000 and reverts with LimitExceeded, blocking a swap that respects the cap. So legitimate redemptions are rejected whenever the funding path deallocates the same account, degrading availability of the redemption facility.
Recommendations	Do the limit check on the balance after deallocation, if deallocation happens.
Comments / Resolution	Fixed.

Issue_17	Midas redemption fee not incorporated
Severity	Medium
Description	Midas protocol skims its fee in mtokens and credits the account only amountMTokenWithoutFee. So the oracle must take into account this fee for withdrawals when calculating the price of the conversion, or the vault will constantly lose value. Furthermore, the TOKEN_TO_REDEEM tokens in the account contracts are valued at the full gross oracle rate with no fee term, while the realization records value net of the Midas fee (amountMTokenWithoutFee). Thus, the assets in the account are not valued correctly and overvalued since the entire value cannot be recovered due to the withdrawal fee.
Recommendations	Incorporate the midas redemption fee into the oracle feeds for the tokens.
Comments / Resolution	Acknowledged.

Issue_18	Discount swaps can be sandwiched for profit
Severity	**Medium**
Description	Discount swaps are profitable for the vault. This is because more dollar-denominated funds are taken in than given out. Also, this profit happens on a per-transaction basis, and not dripped over time. This means any user can make a large deposit to the vault, carry out a discount swap, and then immediately withdraw their funds and capture a large part of the profit generated from the swap. They basically recover a part of the discount that they paid for the swap. This affects the other vault participants as well, since they were expecting profit from a large transaction, but most of those proceeds are captured by a single user. Furthermore, due to the lack of checks present in the Adapter contract described in other issues, any user can trigger a discount swap, bypassing the protocol and the Reactor. So a user can make a large deposit to the vault. Then they can send the tokens to redeem to this adapter and call swap with valid signatures, carrying out the swap. They can then withdraw from the vault, keeping most of the discount that they paid, denying other vault shareholders. All these transactions happened without any input from the protocol or the Reactor/Executor contracts.
Recommendations	Consider adding a delay for withdrawals, or dripping the discount proceeds to the totalAssets of the shareholders over time.
Comments / Resolution	Acknowledged.

Issue_19	A single frozen or blacklisted redemption account freezes deallocation and all vault withdrawals
Severity	Low
Description	deallocate loops over the accounts and safeTransferFroms each one's asset balance to the adapter, with no try/catch and no per-account isolation; every Account grants the adapter an unbounded approval. If the asset is freeze-capable (e.g. USDC) and the issuer blacklists one account, or the token is paused, that single transfer reverts, and the entire withdrawal chain (sweepPending/_deallocate/_deallocateAll/onWithdraw → VaultV2.withdraw/redeem) reverts, blocking withdrawals for every depositor, not just the affected one.
Recommendations	Consider acknowledging and documenting this issue.
Comments / Resolution	Acknowledged.

Issue_20	Liquidlane limits are enforced on discounted output instead of full account exposure
Severity	Low
Description	LiquidLane enforces the per-token limit against the vault asset amount paid out to the swap recipient, after applying the configured discount. The account exposure created by the swap can be larger than that discounted output, because the token to redeem input is worth more in vault asset terms than the discounted amountOut. For example, if the vault asset is USDC and the token being redeemed is worth 100 USDC, a discounted swap might pay only 95 USDC to the filler. The limit check only adds the 95 USDC output to account totalAssets, even though the account receives token exposure worth around 100 USDC before discount. This means the economic exposure to a token-to-redeem market can exceed the configured limit by the discount amount.
Recommendations	Enforce limits against the full oracle value of the token-to-redeem amount entering the account, not only the discounted vault asset output. Alternatively, document that per-token limits are discounted-output limits and set operational limits low enough to account for the maximum discount headroom.
Comments / Resolution	Acknowledged.

Issue_21	LiquidLane signed swap nonces are global per token across signers and quote types
Severity	Low
Description	LiquidLane signed swap nonces are scoped per token rather than per signer and per quote type. A valid signed swap or discount flow can consume a nonce that another signer or quote type expected to use for the same token. This can cause otherwise valid quotes to revert or become unusable. It is not a direct theft path, but it can create execution failures, operational confusion, and denial of service for offchain quoting systems that assume nonces are independent across signers or quote types.
Recommendations	Scope signed swap nonces by signer and quote domain, or document that nonces are global per token and must be coordinated by the offchain quoting system. Consider using separate nonce spaces for delegated swaps and discount policy swaps.
Comments / Resolution	Acknowledged.

Issue_22	Account asset dust donations can grief exact headroom liquidlane swaps
Severity	Low
Description	LiquidLane checks the per token limit by adding the account's current totalAssets to the amount being allocated. Because account totalAssets can include the account's direct vault asset balance, a third party can donate a small amount of the vault asset to the account. If a swap is sized to use the exact remaining limit headroom, even a one unit donation can make the limit check fail. This can grief otherwise valid swaps until the dust is swept or the quote is resized.
Recommendations	Consider acknowledging this issue.
Comments / Resolution	Acknowledged.

Issue_23	Pause mode does not cover account sync or cow conversion
Severity	Low
Description	LiquidLaneAdapter.pause blocks swaps through the whenNotPaused modifier on the internal swap path. It does not pause account level sync or CoW conversion functions. This means that while swaps are paused, external callers can still interact with active accounts by calling sync or by using the account's conversion surface where permitted. These calls can finalize requests, create new issuer redemption requests when cooldown permits, or pre-sign CoW orders for eligible account held tokens. This may be intended if the pause mode is only meant to stop new swaps while allowing settlement and cleanup. However, if operators expect a pause to freeze all market changing account activity, the current behavior leaves important side effects live.
Recommendations	Clarify the scope of pause in documentation and operational procedures. If emergency pause should stop account side effects as well as swaps, add pause checks to account sync and conversion paths or add a separate market quarantine mode that disables new request creation and CoW order signing while still allowing controlled recovery.
Comments / Resolution	Acknowledged.

Issue_24	Discount swap lacks a caller binding, allowing anyone holding the signatures to submit it
Severity	Low
Description	While the Reactor contract ensures that only specific fillers can call swap, there are no such protections at the LiquidLaneAdapter level. Thus, anyone can potentially get access to valid signatures in-flight and submit them to this contract to trigger a swap. The impact is limited, since funds are not transferred in this contract and is instead pre-funded. However, signatures can still be used by unauthorized users as long as they pre-fund the contracts as well.
Recommendations	Consider checking the allowed filler in the swap function as well.
Comments / Resolution	Acknowledged.

Issue_25	protocolSignature can be faked by submitter
Severity	**Low**
Description	In the discount swap function, a protocol signature is required for a valid swap. *if (!SignatureChecker.isValidSignatureNow(* *discount.protocol,* *_hashTypedDataV4(...)* *)* *),* *protocolSignature* *)) {* *revert InvalidSignature();* *}* However, this protocol signature can be easily faked, since discount.protocol is never checked against a whitelist. Thus, a caller with a valid DiscountSwap struct can just select any discount.protocol address that they control, and then generate the corresponding protocolSignature. This enables callers to bypass any verification from the protocol, since they can generate their own protocol signatures.
Recommendations	Consider enforcing a whitelist for protocol addresses or acknowledging this issue.
Comments / Resolution	Acknowledged.

Issue_26	Swap amount and recipient are not bound in the signature
Severity	**Low**
Description	In a discount swap, the caller specifies the recipient address and amountIn. These values are not used in any of the signatures, and thus can be arbitrarily chosen by the caller. Thus, a caller can submit a valid discount swap with an inflated amountIn, leading to a loss. This is partially mitigated by the controls present in the Reactor contract, only allowing specific fillers, etc., but contract-level checks in the Adapter contract are missing. This is in contrast to the other swap functions, which have all their inputs in the signature. If a filler contract different from Reactor is used, this lack of checks can lead to major issues.
Recommendations	Consider bounding the input amount to the signature as well.
Comments / Resolution	Acknowledged.

Issue_27	removeTokenToRedeem can be easily griefed
Severity	Low
Description	In the removeTokenToRedeem function, the contract first ensures that no assets exist in that account. *if (IAccount(accounts[tokenToRedeem]).totalAssets() > 0) {* *revert AccountHasAssets();* *}* This check, however, can be easily griefed, since anyone can donate a single wei of funds to the account and block this, since the totalAssets function also takes into account the balance in the contract.
Recommendations	Consider adding a time-delayed force-removal option.
Comments / Resolution	Acknowledged.

Issue_28	Nonce not used up in discount swaps
Severity	Informational
Description	In discount swaps, the nonce value is never consumed. Unlike the normal swap functions, isUsedNonce is never called. The nonce is consumed in the Reactor contract, preventing replay attacks. However, if a different filler contract is ever used, this can lead to major issues, leading to replaying of signatures.
Recommendations	Consume the nonce in the discount swap function as well.
Comments / Resolution	Acknowledged.

Issue_29	Unsafe deallocation can lead to a drop in totalAssets
Severity	**Informational**
Description	In deallocate, the funds are deallocated to the Adapter itself. *IERC20(asset).safeTransferFrom(account, address(this), amount);* The totalAssets function of the adapter, however, does not take into account its own balance. It only adds the balances of the accounts. Thus, any funds brought from the account to the adapter itself do not register in the totalAssets of the vault anymore. This issue is mitigated by the UniversalDelegator, which always calls Vault.push on any deallocated funds. If this push call is ever skipped, however, the totalAssets of the vault can decrease since even though the vault holds the funds, it does not register in totalAssets anymore.
Recommendations	Consider documenting and acknowledging this. All deallocations must be followed by a vault push, otherwise funds can go unaccounted for.
Comments / Resolution	Acknowledged.

Issue_30	Prefunded direct swap inputs can be claimed by another authorized caller
Severity	Informational
Description	Direct LiquidLane swaps do not bind the tokenIn funding to a specific caller. If tokenIn is prefunded to the adapter before execution, another authorized caller can execute a compatible swap and use that prefunded balance as the input. This lets one authorized caller claim value that another filler or integration intended to use for its own swap. The attacker still needs to be in an authorized execution set, such as the owner, market maker, or an approved filler, but the issue can cause a direct value transfer between authorized participants. The risk is highest if integrations pre-send tokenIn to the adapter before calling the swap function instead of transferring atomically as part of a controlled execution path.
Recommendations	Avoid prefunding swap inputs to the adapter. Consider pulling tokenIn from msg.sender during swap execution, or binding prefunded balances to a caller or quote identifier.
Comments / Resolution	Acknowledged.

Issue_31	Partial acquisition rounding can spend acquisition balances without delivering token units
Severity	Informational
Description	LiquidLane can split tokenIn acquired from a swap between the curator and market maker based on acquisition balances. The split uses proportional math and rounding. In edge cases, acquisition balance can be debited even though the corresponding tokenIn amount rounds down to zero or does not deliver a meaningful token unit. For the tokens currently expected to be used, this is mostly a dust level issue because normal decimals make the rounding loss small. It becomes more relevant for low decimal tokens, very small swaps, or repeated dust-size fills.
Recommendations	Consider acknowledging this issue.
Comments / Resolution	Acknowledged.

Issue_32	Callback-capable LiquidLane assets can reenter while the swap and deallocation state is incomplete
Severity	Informational
Description	Several LiquidLane paths transfer tokens or vault assets before all related accounting is fully settled. If a configured asset has callback behavior, such as ERC777 style hooks, that callback can reenter related protocol paths during the intermediate state.
Recommendations	Document that callback-capable tokens are unsupported, or add reentrancy protection before supporting them.
Comments / Resolution	Acknowledged.

Issue_33	getMaxAssets can mutate delegator state
Severity	Informational
Description	getMaxAssets is not a view function and calls sweepPending before returning capacity. Callers expecting a harmless quote can therefore trigger withdrawal queue or adapter synchronization side effects.
Recommendations	Document getMaxAssets as state changing. Furthermore, consider adding a separate view only quote function.
Comments / Resolution	Acknowledged.

Account

Account is the base redemption-account contract for token-to-redeem integrations. It combines migratable ownership, CoW conversion support, token valuation, and a common totalAssets view consumed by LiquidLaneAdapter, UniversalDelegator, and VaultV2 accounting.

Each account is bound to one immutable TOKEN_TO_REDEEM and one immutable oracle. During initialization it stores the vault and adapter, sets the vault asset and asset unit, configures the initial CoW converter as the account owner, and grants the adapter maximum allowance for the vault asset so realized proceeds can be deallocated.

totalAssets counts three buckets: direct vault-asset balance held by the account, raw token-to-redeem balance valued through the oracle, and child-contract-specific assets returned by _totalAssets. sync is permissionless and nonreentrant, delegating to child _sync.

convert is the account-level CoW conversion path. It first calls sync, then allows conversion only from non-vault-asset and non-token-to-redeem balances into the vault asset. The inherited CoWSwapConverter creates the CoW order with the account itself as receiver, so converted assets return to the account and can later be swept by the adapter.

Appendix: Valuation

Token-to-redeem balances are converted to vault asset units with an oracle rate in 1e18 precision and both token decimal systems. Redemption token helper valuation only normalizes decimals and assumes the child integration has selected a token whose unit is compatible with the vault asset.

Privileged Functions

- setConverters
- invalidateConvert

Invariants

- totalAssets must equal direct vault-asset balance plus oracle-valued token-to-redeem balance plus child _totalAssets.
- Token-to-redeem oracle price must be nonzero before raw token balances are valued.
- convert must not sell the vault asset or the token-to-redeem, and must buy only the vault asset.
- sync must run before any account-level conversion order is created.
- CoW conversion output must return to the account contract.
- The adapter must retain allowance to pull realized vault-asset balances during deallocation.

- Child accounts must implement _sync and _totalAssets without double-counting or omitting assets already handled by the base account.

Issue_34	Fallback redemption tokens are counted at decimal par
Severity	**Governance**
Description	Fallback redemption tokens are valued by decimal normalization only, without an oracle or slippage check. If a fallback token is not actually equivalent to the vault asset, account NAV can overstate the value that can be recovered.
Recommendations	Only configure fallback redemption tokens that are intended to trade at par with the vault asset. If non-par fallback tokens are supported, price them with an oracle or conservative haircut before including them in NAV.
Comments / Resolution	Acknowledged.

Issue_35	Performance fees can crystallize on unrealized external account marks
Severity	**Low**
Description	Account.totalAssets can include external account marks before the value has been realized into the vault asset. For example, held redemption tokens, pending redemption requests, or intermediate claim tokens can be priced through the account oracle and included in LiquidLaneAdapter.totalAssets before the assets are actually available for vault withdrawals. VaultV2 fee accounting treats increases in adapter-reported totalAssets as performance. This means a positive mark from an external issuer or oracle can mint performance fee shares even though the account has not yet received withdrawable vault assets. If the marked value later settles lower, becomes delayed, or cannot be redeemed, existing vault holders are diluted by fees on value that was not actually realized. This is most relevant for integrations such as Midas, where redemption requests can remain pending while their value is still included in account NAV. If performance fees are meant to apply to every mark-to-market increase, this is expected behavior. If they are meant to apply only to realized vault asset gains, the current accounting is too early.
Recommendations	Consider charging performance fees only on realized vault-asset gains, or separate realized assets from externally marked pending value in the adapter accounting exposed to VaultV2. Alternatively, document that performance fees can be minted on unrealized account marks and that later settlement risk remains with vault shareholders.
Comments / Resolution	Acknowledged.

Issue_36	A reverting Midas data feed bricks all vault accounting and cannot be removed or repointed
Severity	Informational
Description	Account.totalAssets values held TOKEN_TO_REDEEM through MidasOracle.getPrice → DataFeed._getDataInBase18, which hard-reverts on staleness (timestamp – updatedAt > healthyDiff), out-of-bounds answers, or non-positive values. The revert is never caught: it propagates through the adapter and delegator NAV loop-sums into VaultV2.getAccrueInterest, which every deposit, withdraw, redeem, and pull calls. One unhealthy feed therefore bricks the whole vault. There is no fallback (no last-good price, no oracle-free valuation branch) and no escape. removeTokenToRedeem and removeAdapter read the same reverting totalAssets, pause is swap-only, and ORACLE/DATA_FEED are immutable, so the feed cannot be repointed or the token removed. The trigger is external (the Midas feed reaching its documented staleness, deprecation, or out-of-bounds state) rather than an attacker, but the consequence is an escape-resistant freeze of all depositor funds.
Recommendations	Consider wrapping oracle reads in try/catch with a defined degraded-mode behavior (last-good price within a staleness bound, or excluding the unreadable token from the sum) so one feed cannot brick global NAV. Make removeTokenToRedeem/removeAdapter robust to a reverting totalAssets so a bad feed can always be decommissioned. Allow governance to repoint the oracle or replace the account for an existing token. Otherwise, consider documenting this issue as an external integration risk.
Comments / Resolution	Acknowledged.

AccountRegistry

AccountRegistry is the owned registry that maps a vault asset and token-to-redeem pair to the account factory that LiquidLaneAdapter should use when onboarding that token. The registry is a configuration boundary: LiquidLaneAdapter trusts the returned factory when it creates token-specific accounts.

Entries are one-way. Once a factory is set for an (asset, tokenToRedeem) pair, it cannot be replaced through this registry. The contract does not perform local validation that the factory is nonzero, compatible with the asset, compatible with the token, or safe to use; those checks must be handled operationally or by the consuming adapter/factory system.

Privileged Functions

- setAccountFactory

Invariants

- Each (asset, tokenToRedeem) pair can be assigned at most once.
- The owner is the only account that can register account factories.
- LiquidLaneAdapter account creation depends on registry entries matching the expected asset, token-to-redeem, account type, and factory version.
- A registered factory should remain valid for all future account creations because the registry cannot update it after the first set.

Issue_37	Account factory selection is not post-validated and can break liquidlane account invariants
Severity	Governance
Description	AccountRegistry controls which factory LiquidLane uses for each asset and token to redeem pair. If governance registers the wrong factory, LiquidLane can create an account with unexpected token, vault, adapter, owner, converter, or accounting assumptions. The registry entry is one-way, account creation uses the factory's latest version, and the created or migrated account is not strongly post-validated. This is a governance configuration risk, but a bad account can break token-specific accounting or route conversions through unexpected logic.
Recommendations	Consider acknowledging this issue.
Comments / Resolution	Acknowledged.

MidasAccount

MidasAccount is the Midas-specific redemption account base. It extends CooldownAccount, tracks Midas redemption request ids, approves the Midas RedemptionVault to pull the account's token-to-redeem, and reports pending requests plus fallback redemption-token balances into account totalAssets.

When _requestRedeem runs, the account checks whether the Midas RedemptionVault has a data feed configured for the vault asset. If it does, the account requests redemption into the vault asset. If not, it requests redemption into the configured fallback REDEMPTION_TOKEN. The request is submitted by the account itself, so Midas fulfillment sends the requested output token back to the account.

_finalizeRequests removes any tracked request whose Midas status is no longer pending. Pending request valuation depends on the concrete child: MidasCompAccount aggregates pending mToken amounts and prices them at the current oracle rate, while MidasNonCompAccount values each pending request using the stored request rate returned by Midas.

Fallback redemption-token balances are counted by _totalAssets when REDEMPTION_TOKEN != _asset. They are not directly sweepable by LiquidLaneAdapter deallocation until converted into the vault asset, typically through the inherited CoW conversion path.

Appendix: Live Orders

Midas request ids are pushed after redeemRequest returns. The Midas vault stores the account as the request recipient/sender, transfers output to the account on approval, and marks the request processed or cancelled externally. Local cleanup occurs on later sync through _finalizeRequests.

Privileged Functions

None directly. MidasAccount inherits account ownership and converter administration from Account/CoWSwapConverter.

Invariants

- requestIds should represent requests that may still be pending until _finalizeRequests removes terminal requests.
- Pending requests must be valued according to the selected compounding or non-compounding account mode.
- Fallback REDEMPTION_TOKEN balances must be counted only when they are distinct from the vault asset.
- New redemption requests must use the full current TOKEN_TO_REDEEM balance held by the account.
- Requested Midas output token must be the vault asset when Midas supports it, otherwise the fallback redemption token.
- Fulfilled Midas output must arrive at the account and become either direct vault asset or fallback-token value awaiting conversion.
- The Midas RedemptionVault allowance for TOKEN_TO_REDEEM must remain sufficient for request creation.

Issue_38	Midas redemption allowances can strand submitted requests
Severity	Governance
Description	Midas requests can be accepted at creation but later depend on external allowance and policy checks for approval. MidasAccount submits the account's full TOKEN_TO_REDEEM balance when creating a request, so unsolicited token donations can also increase the submitted amount. If LiquidLane limits exceed the effective Midas redemption allowance, or if a donation pushes the account balance above the remaining allowance, pending requests can remain counted in NAV while they cannot currently be processed.
Recommendations	Keep LiquidLane limits aligned with external Midas redemption allowances. Consider checking available allowance before creating new requests or stopping swaps when pending requests approach the external cap.
Comments / Resolution	Acknowledged.

Issue_39	Pending midas request marks can diverge from settlement rates
Severity	Medium
Description	MidasAccount values pending redemptions before the external Midas RedemptionVault has settled them. The local mark must therefore match the rate basis that Midas will actually use at approval time. MidasCompAccount marks pending requests at the current oracle rate, while MidasNonCompAccount marks them from the request's stored mTokenRate. Midas settlement can use a live rate, a saved request rate, or an admin supplied approval rate depending on the approval path. If the account type and Midas approval policy do not match, pending request NAV can be higher or lower than the value that will actually be paid. The pending valuation also does not account for the request's payout token and tokenOutRate in all cases. For fallback payout tokens, this can create a second mismatch between the value counted while the request is pending and the value counted once the payout token arrives in the account. Vault deposits, withdrawals, and fee accrual can therefore use a distorted pending value until the request is synchronized or settled.
Recommendations	Value pending Midas requests using the same rate basis and payout token formula that Midas will use for settlement, including tokenOutRate when applicable. Pair each account type with the matching Midas approval policy. Use MidasCompAccount only where live rate marking matches the expected approval path, and use MidasNonCompAccount only where saved request rate marking matches the expected approval path. Avoid charging performance fees or allowing sensitive share price actions from pending marks that can diverge from final settlement value.
Comments / Resolution	Fixed. But the broader dependency on matching account type to Midas settlement policy still remains.

Issue_40	Redemption token price not incorporated
Severity	**Medium**
Description	In Midas Account, sometimes the redeemed tokens are not the same as vault tokens. Still, their value is accounted for in the totalAssets function. *function _totalAssets() internal view override returns (uint256 assets) {* *if (REDEMPTION_TOKEN != _asset) {* *assets += _redemptionTokenToAssets(REDEMPTION_TOKEN, IERC20(REDEMPTION_TOKEN).balanceOf(address(this)));* *}* *assets += _pendingAssets();* *}* The issue is that _redemptionTokenToAssets only shifts the decimals of the REDEMPTION_TOKEN and doesn't actually convert based on price. So REDEMPTION_TOKEN amount is added to the actual vault asset token amount, and is assumed to be of equal value. However, if the price isn't the same, this can lead to issues. If the REDEMPTION_TOKEN is USD and the vault asset is USDC, then any small depeg in USDC will undervalue the vault, since USD and USDC are not the same. Similarly, if REDEMPTION_TOKEN is some other stablecoin with different decimals, and a difference in price between that and USDC is not accounted for here, it can lead to jumps/dips in price during conversions.
Recommendations	Consider incorporating an oracle to price everything in vault assets.
Comments / Resolution	Acknowledged.

Issue_41	Rejected midas redemptions are written down when synced
Severity	**Low**
Description	MidasAccount counts pending Midas redemption requests in account NAV, while the external RedemptionVault reports them as pending. When a request becomes non-pending, the account removes the request ID from local accounting. For processed requests, this is expected because the account should have received the redemption output. For rejected or canceled requests in the verified Midas RedemptionVault, the mTokens are not returned to the account. The account therefore, has no local asset to keep counting, and the pending value is written down when sync or totalAssets observes the final status. This is mainly an external integration risk. A Midas rejection can cause an abrupt NAV decrease, but the lost value is no longer held by MidasAccount and cannot be recovered by local accounting alone.
Recommendations	Document that rejected Midas redemptions are treated as realized external losses once synchronized. Alternatively, consider not removing canceled request IDs from requestIds until the account confirms the mToken was returned or the position is explicitly written off through a controlled admin path.
Comments / Resolution	Acknowledged.

Issue_42	Unsettled midas request arrays can make account nav expensive potentially running out of gas
Severity	Low
Description	MidasAccount stores submitted redemption request IDs until sync observes that they are no longer pending. Account valuation then iterates over the active request list. If requests are created faster than Midas settles them, LiquidLaneAdapter.totalAssets can become increasingly expensive because it includes account totalAssets. This is not permanent growth, since processed requests can be pruned. The concern is temporary gas and liveness pressure while many requests remain pending.
Recommendations	Bound the number of simultaneously pending Midas requests per account, aggregate requests when possible, or use a keeper path to finalize old requests before account's NAV becomes too expensive to read.
Comments / Resolution	Acknowledged.

Issue_43	Immutable redemption token cannot track Midas payment token removal
Severity	Low
Description	MidasAccount sets REDEMPTION_TOKEN as immutable at deployment. Midas manages the same token dynamically in RedemptionVault._paymentTokens, and a vault admin can remove it with removePaymentToken. On redemption, the account picks the output token from tokensConfig for the vault asset: no data feed means it falls back to REDEMPTION_TOKEN, otherwise it uses the vault asset. It never checks that the chosen token is still in Midas's payment token set before calling redeemRequest. If Midas removes that token later, RedemptionVault reverts in _requireTokenExists. The account cannot update REDEMPTION_TOKEN or switch to another registered token. CooldownAccount._sync redeems whenever the account holds TOKEN_TO_REDEEM and cooldown (or owner) allows it. A failed redeemRequest reverts the whole sync, including request finalization. convert and swap flows that call sync fail while mToken remains on the account. Funds are not stolen, but redemption stalls until Midas restores the token or the integration is upgraded. And if the token support is never restored, any TOKEN_TO_REDEEM left in the MidasAccount is stuck indefinitely.
Recommendations	Before calling redeemRequest, confirm the output token is still registered in Midas. If not, skip submission or use another valid token instead of reverting. Change the TOKEN_TO_REDEEM from an immutable variable to a storage variable that can be updated by the MidasAccount owner.
Comments / Resolution	Acknowledged.

CooldownAccount

CooldownAccount is the base account layer for integrations where redemption request creation is cooldown-gated. It provides a shared _sync implementation that first finalizes old requests, then decides whether the current raw token-to-redeem balance can be submitted into a new external request.

A new request is created only when the account has a positive token-to-redeem balance and one of three conditions is true: the caller is the account owner, no request has ever been made, or the cooldown period has elapsed since the last request. When a request is created, lastRequestTimestamp is updated to the current timestamp.

If cooldown has not elapsed and the caller is not the owner, token-to-redeem can remain idle in the account. That idle balance is still handled by the base Account accounting path and valued through the token-to-redeem oracle.

Privileged Functions

None directly.

Invariants

- _finalizeRequests must run before attempting a new request.
- No request should be created for a zero token-to-redeem balance.
- Non-owner request creation is blocked until the first request or until COOLDOWN has elapsed.
- The owner can force request creation even during cooldown.
- lastRequestTimestamp must update only when a new request is submitted.
- Child implementations must make _finalizeRequests and _requestRedeem consistent with the external issuer lifecycle.

Issue_44	Midas dust can block the conversion of settled fallback redemption tokens
Severity	Medium
Description	Some Midas redemptions can settle into a fallback redemption token instead of the vault asset. The account then needs to convert that token into the vault asset. However, Account.convert() calls sync() before converting the requested fallback token. For Midas accounts, sync() can first try to redeem the account's raw TOKEN_TO_REDEEM balance. If that raw balance is below the Midas minimum redemption amount, the Midas request reverts and the fallback token conversion never starts. This can happen through normal operation, or an external user can grief a conversion attempt by transferring a tiny TOKEN_TO_REDEEM amount to the account before the conversion is executed. LiquidLane acquisition splits can also create this state. If most tokenIn is routed to acquisition receivers and only a below-minimum remainder is sent to the account, the account can be left with a raw Midas balance that blocks the more meaningful fallback token conversion path. If the account only holds the sub-minimum raw balance, the impact is bounded to dust-level NAV that is counted but cannot currently move through the normal redemption flow. That can slightly overstate LiquidLane assets or prevent clean token removal until more tokens arrive or an operator handles the dust. This can leave settled fallback redemption tokens stuck in the account while they are still counted as account value. The vault may report NAV that cannot be fully deallocated, and token removal or account cleanup can remain blocked by dust-level balances. The issue is more significant than ordinary dust because a tiny raw Midas balance can block conversion of a much larger already-settled fallback balance.
Recommendations	Add explicit dust handling for raw Midas token balances and fallback redemption tokens.

	The account should be able to convert the meaningful settled fallback amount without first requiring a below-minimum raw Midas balance to be redeemed.
Comments / Resolution	Partially fixed by removing sync call during convert calls. Dust funds still block sync calls however.

Issue_45	Permissionless dust sync can delay redemptions
Severity	**Low**
Description	Account.sync() is permissionless. In CooldownAccount, calling sync() can also start a new redemption request and update lastRequestTimestamp: *if (* *(msg.sender == owner() || lastRequestTimestamp == 0 || block.timestamp >= lastRequestTimestamp + COOLDOWN)* *&& IERC20(TOKEN_TO_REDEEM).balanceOf(address(this)) > 0* *) {* *_requestRedeem();* *lastRequestTimestamp = uint48(block.timestamp);* *}* An attacker can transfer a small accepted amount of TOKEN_TO_REDEEM directly to the account and call sync() when the cooldown is available. This consumes the cooldown with a small redemption request. If a real LiquidLane swap happens shortly after, _swap() sends the meaningful TOKEN_TO_REDEEM amount to the account and calls sync(). However, the caller is the adapter, not the account owner, so it cannot bypass the cooldown. The real redemption may therefore be delayed until the cooldown expires or the owner manually calls sync(). This is for Midas accounts because cooldowns can be long, e.g. 1 day, 3 days for mFONE, and 6 days for mGLOBAL. During the delay, the vault has already paid instant liquidity but the issuer redemption has not started.
Recommendations	Do not let untrusted callers consume the cooldown. Separate permissionless finalization from redemption request creation, or only allow the owner/bound adapter to create new redemption requests.
Comments / Resolution	Acknowledged.

MidasOracle

MidasOracle is a thin oracle adapter for Midas data feeds. It stores one immutable data-feed address and exposes getPrice, returning IMidasDataFeed(DATA_FEED).getDataInBase18() directly.

The returned value is expected to be in 1e18 precision and is consumed by accounts and LiquidLane pricing logic as the token-to-redeem rate. The oracle does not add local freshness checks, zero-value checks, decimal normalization, fallback logic, or try/catch handling.

Privileged Functions

None.

Invariants

- DATA_FEED is immutable after deployment.
- getPrice must return the Midas feed value in 1e18 precision.
- Downstream consumers must handle zero, stale, reverted, or otherwise invalid feed values because MidasOracle forwards the feed result without local validation.

Issue_46	Stale but accepted midas prices can misprice instant redemptions
Severity	**Informational**
Description	LiquidLane uses MidasOracle prices for instant swap validation and account valuation. MidasOracle accepts any price accepted by the external Midas DataFeed. The Midas feed has basic health checks, but the observed mainnet mTBILL feed allows a 30 day healthyDiff and does not enforce a deviation check. This may be too loose if the price is used for instant execution. The impact depends on the Midas feed configuration, token volatility, update cadence, and configured minDiscount.
Recommendations	Document this dependency and consider a stricter local freshness check, deviation check, or token-specific minDiscount buffer for instant swaps.
Comments / Resolution	Partially fixed. Oracle now checks that the value is within a min,max bound, but stale in-range prices still pass.