infiniFi PR 224

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
Slowfi, Security Researcher

December 11, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

infiniFi is a self-coordinated depositor-driven system designed to tackle the challenges of duration gaps in traditional banking.

From Nov 24th to Nov 25th the Cantina team conducted a review of infinifi-contracts on commit hash 135b9ad9. The team identified a total of **18** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	3	1	2
Low Risk	4	2	2
Gas Optimizations	4	3	1
Informational	6	2	4
Total	**18**	**9**	**9**

2.1 Scope

The security review had the following components in scope for infinifi-contracts on commit hash 135b9ad9:

```
src
├── core
│   └── InfiniFiCore.sol
├── integrations
│   └── outland
│       ├── connectors
│       │   ├── ConnectorBase.sol
│       │   ├── ConnectorCCIP.sol
│       │   └── ConnectorLZ.sol
│       ├── Outland.index.sol
│       ├── OutlandFarm.sol
│       ├── OutlandMsgCodec.sol
│       ├── OutlandVault.sol
│       └── portals
│           ├── PortalBase.sol
│           ├── PortalHub.sol
│           └── PortalOutpost.sol
└── libraries
    ├── CoreRoles.sol
    └── EpochLib.sol
```

(files changed in PR 224)

3 Findings

3.1 High Risk

3.1.1 Losses up to liquid buffer do not burn shares, overstating supply

Severity: High Risk

Context: OutlandVault.sol#L151-L154

Description: `OutlandVault.portalUpdate` is intended to align the farm's share balance with the externally reported `_totalAssetsValue`. When a loss is reported it computes `sharesLost = sharesTotalBefore - sharesTotalAfter` and `sharesLiquid = liquidShares()`, then only burns if `sharesLost > sharesLiquid`.

If the loss is less than or equal to current liquidity, the burn branch is skipped and the farm keeps its old share balance. Example: the farm holds 200 shares, the vault has 100 shares worth of liquid assets and `_totalAssetsValue` is reported as 150.

Do notice that this state with higher shares than liquid assets is easily achieved simply after a `portalUpdate` call that reported some earned yield. Continuing the example, here `sharesLost=50` and `sharesLiquid=100` and as `sharesLost` is lower than `sharesLiquid` no burn occurs and the farm remains at 200 shares even though assets are only 150. Accounting that trusts share supply now overstates assets by 50 and redemptions later can hit `InsufficientLiquidity` because outstanding shares exceed backing.

This design prevents readjusting previously recognized assets downward when a loss is within the liquid buffer, so the share supply stays overstated and the reported asset value cannot be corrected for those losses.

Recommendation: On losses, always burn the full delta between `sharesTotalBefore` and `sharesTotalAfter`, capped by the farm balance, without gating on liquidity. This keeps the farm's share supply equal to portal-reported assets while preserving liquidity backing.

infiniFi: Fixed in commit 2a71b182.

Cantina Managed: Fix verified. This change fixes the core bug provided one assumption holds: `_totalAssetsValue` must represent only off-vault/remote assets. Adding `liquidShares()` then sets `sharesTotalAfter` to total (remote + local) assets and burning `sharesTotalBefore - sharesTotalAfter` realigns supply with backing regardless of the buffer, which resolves the issue.

3.2 Medium Risk

3.2.1 Cross-chain pause does not halt portals

Severity: Medium Risk

Context: PortalBase.sol#L14

Description: `PortalHub` and `PortalOutpost` inherit `Pausable` via `CoreControlled`, yet all bridge entrypoints ignore the pause state. `PortalHub.sendTokens` and `PortalHub.receiveMessage` lack the `whenNotPaused` guard. Likewise, `PortalOutpost.sendTokens`, `PortalOutpost.sendAssetsUpdate` and `PortalOutpost.receiveMessage` omit pause enforcement.

Even after `pause()` is invoked by the PAUSE role, keepers and connectors can continue pushing tokens and processing cross-chain messages. This nullifies the intended emergency brake as an incident response can not halt fund movement.

Recommendation: Enforce pause on all externally reachable bridge flows. Add `whenNotPaused` to `sendTokens`, `sendAssetsUpdate` and `receiveMessage` in both portals (and optionally restrict special recovery handlers to `whenPaused`). This lets the PAUSE role stop cross-chain token sends and message handling immediately during incidents.

infiniFi: Fixed in commit 61b0428b.

Cantina Managed: Fix verified.

3.2.2 LayerZero settings are not configured

Severity: Medium Risk

Context: ConnectorLZ.sol#L27

Description: The deployment proposal does not perform any LayerZero OApp configuration, so the `ConnectorLZ` relies on Layer Zero defaults and unset peers. None of the recommended setup calls from the LayerZero docs are executed (`transferOwnership`, `setPeer`, `setEnforcedOptions`, `EndpointV2.setSendLibrary`, `EndpointV2.setReceiveLibrary`, `EndpointV2.setReceiveLibraryTimeout`, `EndpointV2.setConfig` for send/receive, `EndpointV2.setDelegate`). Without these, the OApp will use default libraries/options leading to routing through unintended defaults rather than explicit configuration.

Recommendation: Extend the proposal to explicitly configure the OApp per LayerZero guidance:

- Set the OApp owner/delegate to your multisig wallet.
- Register peers for each destination EID.
- Set enforced options and send/receive libraries (and timeouts/configs) for each path. Choose the right send and receive library in the LayerZero docs.
- Consider adjusting the confirmations respecting the chain's finality.
- Finally, make sure the confirmations set for the send library in Chain A to B, are the same confirmations set for the receive library in Chain B from A.

This ensures deterministic routing and prevents reliance on unspecified defaults.

If you are planning to skip the manual LZ configuration steps (setting the send/receive libraries, confirmations...) consider checking the defaults in LayerZeroScan. However, do note, that these settings are not always up to date.

Relevant documentation link: LayerZero docs on configuring pathways.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.2.3 `portalUpdate` burn logic in `OutlandVault` is sensitive to out of order assets updates

Severity: Medium Risk

Context: OutlandVault.sol#L139-L158

Description: The function `portalUpdate` from contract `OutlandVault` adjusts the farm's share balance so that the total shares match the `_totalAssetsValue` reported by the portal. When `_totalAssetsValue` is lower than the current share supply and the calculated loss exceeds the currently liquid shares, the function burns part of the farm's shares.

This behaviour is correct only if `_totalAssetsValue` is a fresh and ordered view of the total position (local vault liquidity plus remote liquidity). In practice, cross-chain messaging introduces latency and potential reordering, so assets updates can arrive out of date relative to subsequent liquidity-changing actions such as `portalDeposit` and `portalWithdraw`.

The burn path is triggered whenever an update is lower than the current share supply and `sharesLost > sharesLiquid`. That can happen any time there is a mismatch between the ordering/freshness of assets updates and later actions, for example:

Single stale update after profit and withdrawal, turning profit into a loss.

1. Snapshot/report at 100 (message in flight).
2. Profit on L1: `portalDeposit(+60)` → `shares = 160`, `liquidity = 160`.
3. Liquidity sent out before the report lands: `portalWithdraw(140)` → `liquidity = 20`, shares still 160 (no burn here).
4. Stale `portalUpdate(100)` arrives: `sharesLost = 60`, `sharesLiquid = 20`, so it burns `60 - 20 = 40 shares`. Supply drops to 120 while only 20 USDC remain. Part of the profit was treated as a loss solely because the report was stale relative to the intervening mint/withdraw steps.

Similar patterns can occur with multiple updates in flight that are delivered out of order (a higher total applied first, then an older lower total applied later), or with reordering between transfer messages and updates (liquidity bridged out after the snapshot but before the update is processed).

The guard that prevents burning below the current local liquidity (`sharesLost > sharesLiquid`) ensures local backing, but does not guarantee that the resulting share supply remains aligned with the latest intended cross-chain total. As a result, the effective accounting of profit and loss becomes sensitive to message ordering rather than purely to the latest global state.

Recommendation: Consider to make `portalUpdate` resilient to out-of-order or stale assets updates, so that share adjustments always reflect the latest intended total. Possible options include:

- Adding a monotonic nonce or timestamp to assets updates and ignoring or reverting any update older than the last applied one.
- Tracking the latest accepted total and rejecting updates that attempt to move the total to a value inconsistent with known liquidity movements (for example, ignoring a lower total that would retroactively convert previously realized profit into loss).
- Moving from absolute total updates to delta-based updates or a sequenced state machine, so that each update represents a well-ordered change relative to the last applied state.

These changes would reduce the dependency on cross-chain message ordering and help keep the vault's share supply aligned with the protocol's latest cross-chain accounting.

infiniFi: Acknowledged. It is a classic race condition issue. We plan to do these updates every 6-8 hours. In any case message ordering is not enforced and our backend will be configured to never attempt sending new messages until previous ones are delivered.

Cantina Managed: Acknowledged.

3.3 Low Risk

3.3.1 Stale liquidity sync enables MEV on accounting

Severity: Low Risk

Context: OutlandVault.sol#L193-L205

Description: `OutlandVault` only reconciles farm shares to on-chain liquidity when `_syncSharesToLiquidity` (or `portalUpdate`) is explicitly called inside deposit/redeem/portal flows. Between syncs, the farm's share balance stays unchanged while token balances can drift (airdrop/yield/loss), so `OutlandFarm.assets()` and upstream `Accounting.totalAssetsValue()` can be stale. Positive drift understates assets until the next sync mints shares; negative drift overstates assets until a burn. This creates step changes in reported TVL when a privileged actor finally calls `_syncSharesToLiquidity/portalUpdate`.

Because `YieldSharingV2.unaccruedYield()` relies on `Accounting.totalAssetsValue()` vs. `receiptToken.totalSupply()`, a large positive jump allows public users to mint/lock iUSD/siUSD via the gateway right before (or by) calling `accrue()` and capture the pending surplus they did not bear risk for. The accrual mints new receipt tokens to stakers/lockers based on the updated assets, so late joiners can buy cheap to sell at a higher price. Relevant code:

```
function _syncSharesToLiquidity() internal returns (uint256) {
    uint256 sharesLiquid = liquidShares();
    uint256 sharesInFarm = balanceOf(outlandFarm);
    if (sharesInFarm >= sharesLiquid) return 0;
    uint256 sharesToMint = sharesLiquid - sharesInFarm;
    _mint(outlandFarm, sharesToMint);
    return sharesToMint;
}
```

Recommendation: Reduce the step size by syncing frequently (e.g., periodic CCIP/portal updates on a fixed cadence and running the yield smoother), so any jump is small and uneconomic to front-run.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Configurable `chainId` in `PortalBase` can diverge from the actual chain ID

Severity: Low Risk

Context: PortalBase.sol#L18

Description: The contract `PortalBase` defines the state variable `chainId` as a mutable storage variable:

```
uint256 public chainId;
```

and this value is set via initialization logic and used to tag and validate cross-chain messages (for example, when constructing payloads and comparing remote `chainId` values).

Because `chainId` is taken from configuration instead of the EVM's native `chainid` opcode (`block.chainid`), the contract relies on correct governance / deployment configuration to match the actual chain ID. A misconfiguration, upgrade, or faulty initialization could lead to `chainId` being inconsistent with `block.chainid` while the contract continues to operate, which can cause inconsistent identification of the local chain in the Outland protocol's cross-chain messages.

This is especially relevant for portal contracts, which act as trust anchors for routing and validating messages between chains. Having the local chain identifier as a configurable value increases the surface for configuration errors compared to deriving it directly from the execution environment.

Recommendation: Consider to remove the configurable `chainId` storage variable in `PortalBase` and instead derive the local chain identifier directly from the EVM:

- Use `block.chainid` in high-level Solidity wherever the local chain ID is needed.
- Use the `chainid` opcode in inline assembly.

If a stored value is still desired (for gas or interface reasons), consider to:

- Set `chainId` once using `block.chainid` in the initializer.
- Make it effectively immutable (no external setter).
- Optionally add an internal sanity check that `chainId == block.chainid` before using it in critical message-processing paths.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Missing events on configuration and asset-management state transitions

Severity: Low Risk

Context: ConnectorLZ.sol#L62-L68

Description: The functions that modify protocol configuration and asset routing in the Outland bridging stack do not consistently emit dedicated events. In particular, the following functions perform state transitions without emitting an event:

- The function `setConfiguration` from contract `ConnectorBase` updates the `chainConfig` / `selectorConfig` mappings used for cross-chain routing, but does not emit an event when configuration changes.
- The functions `enableChainAsset` and `disableChainAsset` from contract `ConnectorBase` update the per-chain supported asset set, but do not emit any event on asset enable/disable.
- The internal function `_setConfiguration` from contract `ConnectorBase` mutates the same configuration mappings and is used by higher-level configuration functions, but does not emit an event.
- The function `pullAssets` from contract `ConnectorCCIP` transfers tokens from the connector to the portal without emitting any connector-specific event (only the ERC20 `Transfer` event is available).
- The function `setConfiguration` from contract `ConnectorLZ` updates the cross-chain configuration and LayerZero peers, but does not emit an event.
- The functions `enableOFT` and `disableOFT` from contract `ConnectorLZ` update the `ofts` registry for asset → OFT mapping, but do not emit events when the mapping is added or removed.

- The function `pullAssets` from contract `ConnectorLZ` transfers tokens from the connector to the portal without emitting any connector-specific event.
- The function `addConnector` from contract `PortalBase` adds a new connector address to the authorized set, but does not emit an event.
- The function `removeConnector` from contract `PortalBase` removes a connector from the authorized set, but does not emit an event.
- The function `init` from contract `PortalHub` sets the core address and hub chain identifier, but does not emit an event.
- The function `setVault` from contract `PortalHub` registers or updates the `OutlandVault` associated with a chain, but does not emit an event.
- The function `setAssetMapping` from contract `PortalHub` updates the bidirectional mapping between hub assets and outpost assets for a chain, but does not emit an event.
- The function `init` from contract `PortalOutpost` sets the core, local chain identifier, hub chain identifier, accounting contract, farm registry, and receiver farm, but does not emit an event.
- The function `setReceiverFarm` from contract `PortalOutpost` updates the `receiverFarm` used to receive bridged assets, but does not emit an event.

These functions control critical configuration and asset-routing behaviour for the bridging system. Without explicit events, off-chain monitoring, auditing, and incident response have to rely on low-level storage inspection or indirect effects, which makes it harder to track configuration changes, asset whitelist changes, and updates to routing endpoints over time.

Recommendation: Consider to introduce explicit events for all external (and relevant internal) state transition functions that change configuration or asset routing, including the ones listed above. For example, events can capture the previous and new values for:

- Chain configuration (peer, selector, gas limit) in `ConnectorBase` and `ConnectorLZ`.
- Enabled / disabled chain assets in `ConnectorBase`.
- Enabled / disabled OFTs in `ConnectorLZ`.
- Added / removed connectors in `PortalBase`.
- Registered vaults and asset mappings in `PortalHub`.
- Initialization and updates of `receiverFarm` and other key parameters in `PortalHub` and `PortalOutpost`.
- Asset pulls from connectors (`pullAssets`), if you want explicit logs for cross-component token flows in addition to ERC20 `Transfer` events.

This would provide clearer on-chain observability for operational changes, simplify indexer implementations, and help detect misconfiguration or unexpected changes more easily.

infiniFi: Fixed in commit e4753585.

Cantina Managed: Fix verified. It adds some events (`ConfigurationSet`, `AssetEnabled/Disabled` in `ConnectorBase`; `OFTSet/Disabled` in `ConnectorLZ`; `VaultSet` and `AssetMappingSet` in `PortalHub`; `ReceiverFarmSet` in `PortalOutpost`), but the original issue isn't totally fixed as:

- `ConnectorBase.pullAssets` (used by CCIP and LZ) still emits no connector-specific event, so asset pulls remain unlogged beyond ERC20 Transfers.
- `PortalHub.init` still emits nothing for core/hub chain setup.
- `PortalOutpost.init` still emits nothing for core/chain/accounting/farm registry/receiver setup.

3.3.4 Missing storage gap in upgradeable base contract `PortalBase`

Severity: Low Risk

Context: PortalBase.sol#L22

Description: The contract `PortalBase` declares state variables (`chainId` and the `connectors` set) and is intended to be used in an upgradeable setup (through a proxy), but it does not reserve a storage gap at the end of its storage layout.

When an upgradeable base contract is inherited by other upgradeable contracts, adding new state variables in future versions without a reserved gap increases the risk of storage layout collisions between the implementation and its proxies or between different layers of inheritance. This can lead to silent corruption of state in future upgrades, especially if the inheritance structure changes or additional fields are appended in derived contracts.

Because `PortalBase` is a core shared base for `PortalHub` and `PortalOutpost`, any storage layout mistake here would propagate to all portal implementations.

Recommendation: Consider to add a reserved storage gap at the end of the `PortalBase` storage layout (following the common upgradeable-contract pattern) or to adopt an equivalent structured-storage approach. This would provide room for future variables without shifting the existing storage layout and would reduce the risk of state corruption during upgrades for all contracts inheriting from `PortalBase`.

infiniFi: Fixed in commit 3331ae76.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Cache OFT address in sendTokens to save SLOADs

Severity: Gas Optimization

Context: ConnectorLZ.sol#L103-L139

Description: `ConnectorLZ.sendTokens` repeatedly reads `ofts[_assetToken]` for approval, fee quoting and the send call. Each lookup costs an extra SLOAD. Caching the OFT address once after validation and reusing it would slightly reduce gas for each bridge send. The path is keeper/portal-only, so this is a minor optimization, not a security concern.

Recommendation: Store `address oft = ofts[_assetToken];` after the `require` checks and reuse `oft` for `forceApprove`, `quoteSend` and `send`.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 String-based revert is inconsistent and less efficient

Severity: Gas Optimization

Context: ConnectorLZ.sol#L50

Description: The function `setPeer` from contract `ConnectorLZ` overrides the LayerZero `OApp` hook and unconditionally reverts with a string message indicating that `setConfiguration` should be used instead.

Using a string-based revert here is inconsistent with the rest of the connector contracts, which rely on custom errors for failures. String-based reverts are slightly more gas expensive and less structured than custom errors, and they reduce the ability to programmatically reason about specific failure types. Since this is a deliberate guard rail (to force configuration via `setConfiguration`), it fits well as a dedicated custom error.

Recommendation: Consider to replace the string-based revert in `ConnectorLZ.setPeer` with a custom error that explicitly conveys that direct `setPeer` usage is not allowed and that `setConfiguration` must be used instead. This keeps error handling consistent across the codebase and provides a marginal gas optimization.

infiniFi: Fixed in commit 99ceba64.

Cantina Managed: Fix verified.

3.4.3 Use of `assert` in production code

Severity: Gas Optimization

Context: ConnectorLZ.sol#L213

Description: The function `pullAssets` from contract `ConnectorLZ` uses `assert(msg.sender == portal)` to validate the caller in production code. The functions `init` from contracts `PortalHub` and `PortalOutpost` also use `assert(address(core()) == address(0))` to enforce one-time initialization.

In Solidity $\geq$ 0.8, `assert` is intended for invariants that are assumed to be always true and, on failure, triggers a `Panic(0x01)` and consumes all remaining gas. In the cases above, the conditions depend on deployment or configuration correctness and could realistically be violated due to misconfiguration or misuse. Using `assert` for these checks makes the failure mode less diagnosable and more gas-expensive compared to using `require` with a custom error. It also goes against the common convention of reserving `assert` for internal invariants that the compiler is allowed to treat as unreachable.

Recommendation: Consider to replace the `assert` statements in `ConnectorLZ.pullAssets`, `PortalHub.init`, and `PortalOutpost.init` with `require` statements or custom errors that:

- Explicitly validate `msg.sender` against `portal` in `ConnectorLZ.pullAssets`, and.
- Explicitly enforce the one-time initialization guard in `PortalHub.init` and `PortalOutpost.init`.

This would keep the semantics of the checks while providing clearer error reporting, avoiding `Panic` reverts, and aligning the contracts with common production best practices for error handling.

infiniFi: Fixed in commit 6720af08.

Cantina Managed: Fix verified.

3.4.4 Use of memory for `_data` payload in connector external functions increases gas cost

Severity: Gas Optimization

Context: ConnectorCCIP.sol#L49

Description: The function `sendMessage` from contract `ConnectorLZ` takes its `_data` payload as a bytes value stored in memory, even though this function is an external entry point and only reads the payload before forwarding it. The equivalent functions in the CCIP connector and other Outland components (such as `getMessageFee`, `getSendTokensFee`, and `sendTokens` in both connectors, as well as `receiveMessage` in the portal contracts and the message codec helpers) already operate on the message payload in calldata.

Using a memory-allocated bytes parameter on an external function causes the compiler to copy the entire payload into memory on every call, which increases gas cost without providing additional safety or functionality in this case. Since `_data` is only read and forwarded, keeping it in calldata is sufficient and cheaper. Having one connector function still using memory for the same kind of payload also makes the interface slightly inconsistent between the CCIP and LayerZero connectors.

Recommendation: Consider to update the connector interfaces so that all external and public functions receiving the `_data` message payload use calldata instead of memory, in particular:

- Change the `_data` parameter of `sendMessage` in `ConnectorLZ` to be stored in calldata.
- Align any other external/public message-related functions that still use memory for `_data` (if present in other versions of the connectors) so they also accept `_data` in calldata when the payload is only read.

This would remove unnecessary memory copies, reduce gas consumption on message sends and fee estimations, and keep the connector interfaces consistent across both CCIP and LayerZero implementations.

infiniFi: Fixed in commit d4e502ac.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 CCIP token rate limits unhandled by portals

Severity: Informational

Context: ConnectorCCIP.sol#L19-L21

Description: InfiniFi's CCIP connector and portal flows do not account for Chainlink token pool rate limits. For example, the USDC pool on Base (0x6378c36C44B28f4d1513e7a5510A8481a23eecda, USDCTokenPool) currently has both outbound and inbound rate limiters disabled, so USDC is unaffected. However, any token (including USDC if limits are later enabled) can be throttled at the pool layer:

- Outbound: when a keeper calls `sendTokens` via CCIP, the router invokes `lockOrBurn` on the pool; if the outbound bucket lacks capacity the pool reverts (`TokenRateLimitReached`), preventing `ccipSend` from emitting and causing the entire portal/connector call to revert, burning gas with no state change.
- Inbound: if the pool's inbound bucket is exhausted, the offRamp's `releaseOrMint` reverts on delivery; the message remains pending and tokens stay locked/burned on the source chain until capacity refills, delaying liquidity and skewing `Outland` accounting.

The connectors do not preflight or surface pool headroom, so operators have no visibility and must manually check these limits. Because of this, enabled or tightened pool limits can silently halt outbound sends or stall inbound delivery for any CCIP-listed token; current configs avoid this for USDC, but newly whitelisted tokens must be checked against Chainlink's directory (e.g., USDC: `https://docs.chain.link/ccip/directory/mainnet/token/USDC`; all tokens: `https://docs.chain.link/ccip/directory/mainnet`) to avoid unexpected throttling.

Recommendation: Add operational checks before enabling a token/route on CCIP: query the token pool's `getCurrentOutboundRateLimiterState` and `getCurrentInboundRateLimiterState` for the relevant selectors to confirm limits and headroom, and document the configured rate/capacity for operators. If limits are present, enforce preflight in the keeper workflow (split sends under bucket capacity or wait for refill) and monitor buckets to alert on low headroom. Re-verify these settings whenever a new token is whitelisted or limits are changed.

infiniFi: Acknowledged. This will be monitored off-chain.

Cantina Managed: Acknowledged.

3.5.2 USDeOFT rate limits

Severity: Informational

Context: ConnectorLZ.sol#L24-L27

Description: USDe is bridged via an OFT adapter that enforces per-destination rate limits in `_debit` before a LayerZero send is emitted (`USDeOFT.sol`, `RateLimiter.sol`). Limits are keyed by LayerZero endpoint IDs (EIDs). On Arbitrum, for example, `rateLimits(30101)` returns {`amountInFlight:` 10_008.524852e18, `lastUpdated:` 1764058403, `limit:` 10_000_000e18, `window:` 3600}, meaning about 10k USDe is currently counted against a 10M-per-hour cap that decays over 1h. The limiter is only applied on the send path. `_credit` (receive) is unthrottled, so inbound mints have no rate cap and rely solely on the sender's limiter. The window/limit can also be set to 0, which silently disables throttling or bricks a route. If the rate limit is consumed, Outland Outpost → Hub sends (L2 → L1 via `PortalOutpost` + `ConnectorLZ` + `PortalHub`) will temporary revert with `RateLimitExceeded` before any message is emitted, stranding the bridge action.

Recommendation: Merely informative. Be aware of this behaviour when bridging.

infiniFi: Acknowledged. This will be monitored off-chain.

Cantina Managed: Acknowledged.

3.5.3 Governance receive bypasses xchain auth

Severity: Informational

Context: ConnectorBase.sol#L53-L60

Description: `ConnectorBase.govReceive` lets the `PROTOCOL_PARAMETERS` role push arbitrary `_message` payloads straight to the portal, bypassing the LayerZero/CCIP transport and their peer/chain validation. The inline comment says "should be gated behind 1 day timelock", but no delay or extra authentication is enforced. A compromised or malicious governance signer can immediately forge cross-chain effects (e.g., bogus accounting updates or transfer payloads) without any cross-chain authenticity guarantees, collapsing the trust model to a single role and removing the intended safety window. Because of this, governance compromise or misuse can inject arbitrary portal messages instantly.

Recommendation: Enforce a timelock/multisig delay on `govReceive` (or remove it) and ensure any governance path still validates expected peers/chainIds. Restrict usage to audited emergency flows and emit events for monitoring; prefer routing through the normal transport rather than direct message injection.

infiniFi: Acknowledged. If a chain is unavailable for any reason, something happened, there was a disaster, etc. we will use this method to cast manual updates. Also, if for any reason our bridging partner failed to deliver their message without retry in place, this will be used. It is understandable how this can be abused to create significant losses or inflate the yield. However, doing so is nothing we can profit from other than ruining our reputation. This method is a last resort in case there is anything wrong with a cross-chain communication and is not meant to be used ever in normal circumstances.

Cantina Managed: Acknowledged.

3.5.4 Unused imports and using directives across contracts

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Several contracts carry unused imports/using directives, adding dead code and inflating bytecode/metadata:

- `ConnectorBase` imports `IERC20`, `IERC165`, `SafeERC20`, `IAny2EVMMessageReceiver`, `Client`, `IRouterClient`, and sets `using SafeERC20` but never uses them.
- `ConnectorLZ` imports `CoreControlled` without any reference.
- `OutlandFarm` imports `FixedPointMathLib` and applies `using FixedPointMathLib for uint256` without calling its helpers.

Recommendation: Remove the unused imports and associated `using` statements from these contracts to trim bytecode and eliminate lint noise.

infiniFi: Fixed in commit 99ceba64.

Cantina Managed: Fix verified.

3.5.5 Missing asset mapping check in `receiveMessage`

Severity: Informational

Context: PortalHub.sol#L136-L153

Description: In `PortalHub.receiveMessage` the TRANSFER branch decodes `senderToken` and immediately calls `assetMapping[senderChainId].get(senderToken)` without a `contains` guard. If the mapping is missing, `EnumerableMap.get` reverts with its generic string error instead of the portal's `InvalidAsset` revert used elsewhere:

```
if (messageType == OutlandMsgCodec.TRANSFER) {
    (address senderToken, uint256 amount) = OutlandMsgCodec.getTransferPayload(_data);
    address actualToken = assetMapping[senderChainId].get(senderToken);
    _handleReceiveTokenTransfer(connector, senderChainId, actualToken, amount);
}
```

This yields inconsistent revert reasons and less clear failures for bad/unknown assets.

Recommendation: Mirror the send path: check `assetMapping[senderChainId].contains(senderToken)` and revert `InvalidAsset(senderToken)` before calling `get`. This keeps revert reasons consistent and avoids generic library string reverts.

infiniFi: Fixed in commit 225452a0.

Cantina Managed: Fix verified.

3.5.6 `Outland.index.sol` test-only contract is located under the main source tree

Severity: Informational

Context: Outland.index.sol#L11

Description: The file `Outland.index.sol` under `src/integrations/outland/` appears to be intended only for testing purposes, but it is placed in the main source directory alongside production contracts. Keeping test-only code in the primary source tree can make it harder to distinguish deployable contracts from testing helpers, and can increase the risk that non-production code is accidentally included in builds, audits, or deployment pipelines. It may also introduce noise for tooling that assumes everything under `src/` is part of the production surface.

Recommendation: Consider to move `Outland.index.sol` to a dedicated test directory (for example, `test/` or an equivalent testing-specific folder) or adjust the project structure so that test-only contracts are clearly separated from production contracts and excluded from deployment artifacts.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.