yAudit

June 2026

Prepared for
Paxos Labs

Audited by
watermelon
fedebianu

Paxos Labs Transit Station

Smart Contract Security Assessment

Contents

1 Review Summary

1.1 Protocol Overview

The reviewed Transit Station module implements a 1:1 same-underlying asset order flow where users submit backend-signed quotes, source assets are collected locally, cross-chain order terms are relayed through LayerZero when needed, and a privileged executor fulfills pending orders from a configured want-asset source.

1.2 Audit Scope

This audit covers 6 smart contract totaling approximately 618 lines of code across 3 days of review.

```
src
├── base
│   └── Roles
│       └── CrossChain
│           └── OAppAuth
│               ├── OAppAuthCore.sol
│               ├── OAppAuthReceiver.sol
│               ├── OAppAuthSender.sol
│               └── OAppAuth.sol
├── helper
│   └── Pausable.sol
└── transit
    └── TransitStation.sol
```

1.3 Risk Assessment Framework

1.3.1 Severity Classification

Severity	Description	Potential Impact
Critical	Immediate threat to user funds or protocol integrity	Direct loss of funds, protocol compromise
High	Significant security risk requiring urgent attention	Potential fund loss, major functionality disruption
Medium	Important issue that should be addressed	Limited fund risk, functionality concerns
Low	Minor issue with minimal impact	Best practice violations, minor inefficiencies
Undetermined	Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and although their exact consequences remain uncertain, they present a sufficient potential risk to warrant attention and remediation.	Varies based on actual severity
Gas	Findings that can improve the gas efficiency of the contracts.	Increased transaction costs
Informational	Code quality and best practice recommendations	Reduced maintainability and readability

Table 1: severity classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	5

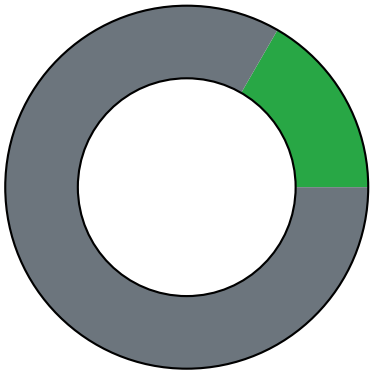

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

No Critical, High, or Medium vulnerabilities were identified under the stated trust model. The review reported one Low severity issue and several Informational findings, primarily around cross-chain liveness, operational recovery, observability, and documentation clarity.

2 Audit Overview

2.1 Project Information

Protocol Name: Paxos Labs
Repository: https://github.com/paxoslabs/nucleus-boring-vault
Commit Hash: 5affe77657a9bf788a6f13c27f911047d8900616
Final Commit Hash: 4a0d21ec5837842cfdb8936816c5e248e5e68c10
Commit URL: https://github.com/paxoslabs/nucleus-boring-vault/commit/5affe77657a9bf788a6f13c27f911047d8900616

2.2 Audit Timeline

The audit was conducted from June 16 to 18, 2026.

2.3 Audit Resources

- Code repositories and documentation

Category	Mark	Description
Access Control	Good	The reviewed contracts uses Solady's Auth contract to define different roles and privileged roles within its contracts. The roles are adequately defined and segregated, such that a single compromise in any low privilege account does not lead to catastrophic consequences. The highest privilege role, which is able to execute critical storage writes and grant or remove other roles, is intended to be assigned to a multisig account.
Mathematics	Good	TransitStation contract uses very simple mathematical relations, particularly when a transit quote is submitted, in order to calculate the maximum amount of fees to be charged.
Complexity	Good	The reviewed contracts present a low degree of complexity: their intended behaviour is clearly defined and accurately implemented. Layerzero's integration poses as a source of complexity, especially revolving the safety of the configuration used once the contracts are deployed.
Libraries	Good	The contracts use the well established Solady library.
Decentralization	Low	TransitStation is a strongly permissioned contract: order submission is gated via ECDSA signatures provided by a proprietary backend, order execution is only executable by accounts with appropriate privilege level and isn't guaranteed.
Documentation	Excellent	Exceptional in-line documentation was provided within the code, along side external development documentation.
Testing and verification	Average	Unit tests for TransitStation are currently being developped within PR 306. The tests cover constructor validation and `submitOrder` failure paths, including route, fee, decimal-conversion, token-transfer and ECDSA signature checks. Furthermore, the protocol team is actively testing the contracts in testnet environments.

Table 2: Code Evaluation Matrix

2.4 Critical Findings

None.

2.5 High Findings

None.

2.6 Medium Findings

None.

2.7 Low Findings

2.7.1 Same-chain order submissions can retain accidental native sent with the transaction

Technical Details

Both submission entrypoints, `submitOrder()` and `submitOrderWithPermit()`, are payable because cross-chain orders use `msg.value` to pay LayerZero fees.
Both functions delegate to `_submitOrder()`. When `quote.route.destEID == thisChainEID`, `_submitOrder()` queues the order locally through `_pushOrder()` and does not call `_sendOrder()`, the only branch that forwards `msg.value` to LayerZero with `msg.sender` as the refund recipient.
The same-chain branch has no `msg.value` check and does not issue a refund. Any native asset accidentally attached to a local submission remains as stray contract balance and can later be swept to the owner through `recoverETH()`, not to the original submitter.

Impact

Low. Users or integrations can accidentally lose native assets attached to same-chain submissions. The loss is limited to accidental `msg.value`, and funds are recoverable only by the privileged owner path.

Recommendation

Reject non-zero `msg.value` for same-chain submissions:

```
1 if (quote.route.destEID == thisChainEID && msg.value != 0) revert UnexpectedMsgValue();
```

Alternatively, immediately refund the full `msg.value` to `msg.sender` before queuing the local order.

Developer Response

Fixed in commit cff03ae.

2.8 Gas Savings Findings

None.

2.9 Informational Findings

2.9.1 Removing a peer does not immediately disable established inbound Transit routes

Removing a destination LayerZero peer is a receive-side quarantine action, not a complete route shutdown. If the source station, source route approval, and backend quote issuance remain active, users can still submit cross-chain orders and have source funds collected before the destination rejects delivery.

Technical Details

Transit uses LayerZero's peer authentication model to control which remote chains can deliver cross-chain orders. The `setPeer(eid, peer)` function in `OAppAuthCore` sets the trusted sender address for a given source endpoint, and `setPeer(eid, 0)` removes that trust relationship. When an operator calls `setPeer(eid, 0)` on the destination chain while the source chain remains active, subsequent cross-chain orders encounter this flow:

1. A user submits a valid quote through `submitOrder()` on the source station
2. `_verifyAndCollect()` immediately transfers the user's offer asset to `protocolFeeRecipient`, `integratorFeeReceiver`, and `offerReceiver`
3. `_sendOrder()` emits a LayerZero packet containing the `OrderTerms`
4. The destination endpoint verifies the packet successfully because the path was already established
5. When execution reaches `lzReceive()` in `OAppAuthReceiver.sol:113-128`, the call to `_getPeerOrRevert(_origin.srcEid)` reverts because the peer mapping now returns zero
6. The revert prevents `_lzReceive()` from queueing the pending order, leaving the user's funds collected on the source chain without a corresponding order on the destination

The user has paid into a route that operators intended to disable. Because the LayerZero delivery reverted atomically, the verified payload remains retryable on the endpoint but cannot be consumed until the peer is restored.

Impact

Informational.

Recommendation

Document peer removal as a coordinated shutdown step rather than a unilateral route kill switch. Operators should first stop new source-side admissions by disabling backend quote issuance and deapproving the affected route on the source station, or by pausing submissions when a broader stop is required. After in-flight LayerZero packets have delivered or been reconciled, the destination peer can be removed safely.

Developer Response

Acknowledged, considering this is assuming misuse we will take the consideration but change nothing in the code.

2.9.2 Cross-chain dust orders can collect source funds without creating a destination order

Technical Details

`TransitStation._verifyAndCollect` validates that the source-chain net amount is nonzero in offer-asset token units before collecting funds from the caller. The method then transfers the protocol fee, integrator fee, and net amount, and returns `offerAmountNormalized18AfterFees` derived from the nominal net amount collected.
For cross-chain orders, `_submitOrder` sends the resulting `OrderTerms` to the destination chain after the source funds have already been collected. The destination later calls `_pushOrder`, which converts `terms.offerAmountNormalized18AfterFees` into want-asset token units and reverts with `ZeroAmountDue` if the conversion truncates to zero.
As a result, an order can pass the source-chain `NetTruncatesToZero` check and still truncate to zero on the destination. For example, if the offer asset has more decimals than the destination want asset, a low-value order may collect a nonzero offer amount on the source while producing zero want-asset units on the destination. In this case the source transaction has already completed, while the destination LayerZero delivery remains retryable and cannot create a pending order without a protocol-side intervention or refund process.

Impact

Informational. Small cross-chain orders may collect user funds without creating a fillable destination order. The issue may be limited by backend quote signing and route approval, but it can still cause avoidable stuck orders and manual refund handling.

Recommendation

Enforce a source-chain minimum that accounts for the destination want asset's decimals before collecting funds. If implementing and maintaining such data structure is deemed to cumbersome, enforcing a minimum order amount for a given want asset can also suffice.

Developer Response

Acknowledged, the backend quote signer may apply minimums effectively and this is not considered an issue as any rounding occurs only in protocol favor and is capped to asset decimals.

2.9.3 `_pushOrder()` has no local idempotency guard for the order UUID

`_pushOrder()` unconditionally writes `pendingOrders[terms.uuid]`, with no check that the UUID is not already pending. UUID uniqueness is therefore guaranteed entirely by two external

invariants rather than by the function itself. Under the current trust model and topology this is sound and not reachable, but a single duplicate push would reset a partially-filled order's `amountDue` back to its full value. A one-line local guard makes the invariant self-contained.

Technical Details

The reason a duplicate `uuid` cannot reach this code today:

1. **Same-chain / source side** — `_verifyAndCollect()` sets `usedDigests[digest] = true` before dispatch, and the `uuid` *is* the digest, so a quote cannot be submitted (and thus pushed/bridged) twice.
2. **Cross-chain receive side** — `_lzReceive()` performs no `usedDigests` check, so it relies solely on the LayerZero endpoint delivering each committed packet at most once.

However, `_pushOrder()` does not enforce that invariant itself. `EnumerableSet.add()` is idempotent and its return value is ignored, while `pendingOrders[terms.uuid] = order` blindly overwrites any existing order.

Impact

Informational.

Recommendation

Make the uniqueness self-contained. `EnumerableSet.add()` already returns whether the element was newly added, so the guard is free:

```
1 if (!pendingOrderIds.add(terms.uuid)) revert OrderAlreadyPending(terms.uuid);
```

Developer Response

Fixed in commit fc99730.

2.9.4 Constructor does not code-check `_authority`

Technical Details

The constructor() verifies `_endpoint.code.length != 0` but performs no code-existence check on `_authority`, although the project's design notes describe a code-existence check on `_authority`.

Impact

Informational.

Recommendation

Align implementation and documentation: add the documented `_authority` code-existence check.

Developer Response

Acknowledged, we perhaps mistakenly mentioned a code-existence check on `_authority` but it should not exist. A 0 address authority is equivalent to using Auth as a simple owner only contract which is desired sometimes even if only during the deployment process.

2.9.5 `recoverETH()` / `recoverTokens()` emit no events

Technical Details

`recoverETH()` and `recoverTokens()` move value out of the contract but emit nothing.

Impact

Informational.

Recommendation

Emit a dedicated event from each recovery function.

Developer Response

Fixed in commit b1d6ac7.

Paxos Labs Transit Station

Completed 2026-06-18