EigenLayer

Multichain Transport

Security Assessment Report

Version: 2.0

August, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the EigenLayer components in scope. The review focused solely on the security aspects of these components, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the components in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the EigenLayer components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the EigenLayer components in scope.

Overview

The EigenLayer multichain protocol distributes stake information across different blockchain networks to support AVS (Actively Validated Services) task confirmation any chain. The protocol enables AVS consumers to verify that operator tasks have been completed by checking this distributed stake data on cheaper L2 networks, rather than requiring verification on the main Ethereum network. This reduces costs while retaining Ethereum mainnet as the source of truth.

The `multichain-transporter` service is a Go-based microservice that distributes stake table data to configured chains. It monitors blockchain events via a Sidecar RPC connection and performs two types of data synchronization: daily transports at 14:00 UTC (updating all operator sets) and *delta* transports triggered by operator set changes (registrations, slashing events). The service uses AWS KMS for transaction signing and BLS signatures for protocol authentication.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to the following commits

- `Layr-Labs/eigenlayer-contracts`: fbfd00c, with the scope restricted to the Solidity contracts in `src/contracts/multichain`.
- `Layr-Labs/multichain-transporter`: 813070e
- `Layr-Labs/multichain-go`: 25557f8

The retesting was performed against the following commits:

- `Layr-Labs/eigenlayer-contracts`: fc1984e
- `Layr-Labs/multichain-transporter`: 40a2a46
- `Layr-Labs/multichain-go`: 788cc7f

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Solidity and Golang.

For the Solidity components, the manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support the Solidity components of the review, the testing team used the following automated testing tools:

- Aderyn: `https://github.com/Cyfrin/aderyn`
- Slither: `https://github.com/trailofbits/slither`

For the Golang components, the manual review focused on identifying issues associated with the business logic implementation of the libraries and modules. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Go runtime.

Additionally, the manual review process focused on identifying vulnerabilities related to known Golang anti-patterns and attack vectors, such as integer overflow, floating point underflow, deadlocking, race conditions, memory and CPU exhaustion attacks, and various panic scenarios including `nil` pointer dereferences, index out of bounds, and explicit panic calls.

To support the Golang components of the review, the testing team used the following automated testing tools:

- golangci-lint: `https://golangci-lint.run/`
- vet: `https://pkg.go.dev/cmd/vet`
- errcheck: `https://github.com/kisielk/errcheck`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 12 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Medium: 1 issue.
- Low: 4 issues.
- Informational: 6 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the EigenLayer components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the components, including optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected components(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
ELM-01	Inconsistent Certificate Signature Schemes Enable Stake Weight Manipulation	High	Resolved
ELM-02	Missing `lastProcessedBlock` Update After Daily Transport Causes Repeated Triggers	Medium	Resolved
ELM-03	TLS MinVersion Too Low - Recommended Security Hardening	Low	Closed
ELM-04	Localhost TLS Bypass	Low	Closed
ELM-05	`BlockNumber` Config Field and CLI Argument Has No Functional Purpose	Low	Resolved
ELM-06	Fragile Logic In `shouldTriggerDailyTransport()`	Low	Resolved
ELM-07	Use Of `Fatalf()` Logging	Informational	Resolved
ELM-08	Direct Error Comparisons Should Use `errors.Is()` For Wrapped Error Detection	Informational	Resolved
ELM-09	Key Access Functions Return Zero Values Without Documentation	Informational	Resolved
ELM-10	`ChainManager` Not Thread-Safe Despite Concurrent Usage Patterns	Informational	Resolved
ELM-11	Incomplete Validation Of `config` Parameter In `_setOperatorSetConfig()`	Informational	Closed
ELM-12	Miscellaneous General Comments	Informational	Resolved

ELM-01	Inconsistent Certificate Signature Schemes Enable Stake Weight Manipulation		
Asset	`BN254CertificateVerifier.sol`, `ECDSACertificateVerifier.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

BN254 and ECDSA certificate verification schemes have inconsistencies in how the `referenceTimestamp` is handled during signature validation, enabling attackers to manipulate stake weights by reinterpreting signatures with different timestamps.

The fundamental issue lies in what data is signed by operators when creating certificates. For ECDSA certificates, operators sign over the complete certificate digest including the `referenceTimestamp` via `calculateCertificateDigestBytes()`. However, BN254 certificates only sign over the `messageHash`, completely excluding the `referenceTimestamp` from the signed data.

In the BN254 verification process, the `_verifySignature()` method only validates signatures against `cert.messageHash`:

```
function _verifySignature(
    BN254Certificate memory cert,
    bytes32 messageHash,
    NonSignerStakesAndSignature memory nonSignerStakesAndSignature
) internal view {
    // Only signs over messageHash, not referenceTimestamp
    BN254.verifySignature(messageHash, nonSignerStakesAndSignature, cert.stakeRegistry);
}
```

Meanwhile, ECDSA certificates include the timestamp in the signed digest:

```
function calculateCertificateDigestBytes(
    ECDSACertificate memory cert
) public pure returns (bytes memory) {
    return abi.encode(
        cert.messageHash,
        cert.referenceTimestamp, // @audit Included in signature
        cert.quorumNumbers
    );
}
```

This inconsistency creates several attack vectors:

- Attackers can construct valid BN254 certificates with arbitrary `referenceTimestamp` values using the same signature set, allowing manipulation of stake weights used for validation
- Historical signatures can be replayed with different timestamps to achieve different quorum thresholds
- The same signature set could validate against multiple stake distributions depending on the chosen timestamp

For long-running tasks, this prevents the system from properly protecting against slashed operators, as BN254 certificates cannot reliably enforce temporal constraints without including the timestamp in the signature.

Recommendations

Modify the BN254 certificate signature scheme to include the `referenceTimestamp` in the signed data, ensuring consistency with the ECDSA implementation. This requires updating both the signature generation and verification logic to sign over a complete certificate digest that includes the timestamp.

Resolution

The development team resolved this vulnerability by updating both certificate verification implementations to ensure consistent handling of the `referenceTimestamp`.

The fix was implemented across two repositories:

- EigenLayer contracts: PR #1610
- Multichain Go implementation: PR #22

The BN254 certificate verification now includes the `referenceTimestamp` in the signed data, preventing stake weight manipulation through timestamp reinterpretation.

ELM-02	Missing `lastProcessedBlock` Update After Daily Transport Causes Repeated Triggers		
Asset	`multichain-transporter: pkg/transporter/transporter.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The `processBlock()` method fails to update `lastProcessedBlock` after successful daily transport completion, causing subsequent blocks to incorrectly trigger additional daily transports.

The problematic control flow occurs at lines [**217-219**]:

```
t.logger.Info("Daily transport completed successfully")
return nil
```

When daily transport completes successfully, `processBlock()` returns early, skipping the `lastProcessedBlock` update at [**236**]. This creates a bug where:

1. Daily transport triggers and completes successfully.
2. `lastProcessedBlock` remains unchanged (pointing to the previous block).
3. The next block processed still passes `shouldTriggerDailyTransport()` because it compares against the stale `lastProcessedBlock`.
4. Daily transport triggers again unnecessarily.

This results in repeated daily transport operations, causing additional gas costs and inefficient resource usage. The issue has high likelihood because it occurs every time daily transport is triggered under normal operations.

The current code only updates `lastProcessedBlock` in the "fall-through" path at [**236**], which is only reached when daily transport doesn't trigger.

Recommendations

Ensure `lastProcessedBlock` is also updated when daily transport succeeds, as well as the delta transport and no-action code paths.

Additionally, implement automated tests that exercise `processBlock()` and verify that `shouldTriggerDailyTransport()` returns false after daily transport has been performed. This will prevent regression and ensure proper state management.

Resolution

The recommendation was implemented in PR #15.

ELM-03	TLS MinVersion Too Low - Recommended Security Hardening		
Asset	`multichain-transporter: pkg/transporter/transporter.go`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The TLS configuration for sidecar RPC connections does not specify a minimum TLS version, defaulting to TLS 1.2. Upgrading to TLS 1.3 provides security hardening against handshake and downgrade attacks.

The current configuration occurs at [**109**]:

```
creds = grpc.WithTransportCredentials(credentials.NewTLS(&tls.Config{InsecureSkipVerify: false}))
```

Without an explicit `MinVersion` setting, Go currently defaults to TLS 1.2. TLS 1.3 provides enhanced security including resistance to protocol downgrade attacks, improved handshake security, and elimination of vulnerable cipher suites.

Recommendations

Update the TLS configuration to enforce TLS 1.3 as the minimum version:

```
creds = grpc.WithTransportCredentials(credentials.NewTLS(&tls.Config{
    InsecureSkipVerify: false,
    MinVersion:         tls.VersionTLS13,
}))
```

This provides enhanced resistance to downgrade attacks and stronger cryptographic guarantees.

Resolution

The issue has been closed as acknowledged. The development team are not exposing the RPC connections publicly and therefore TLS is not a necessary requirement.

> *Since we're talking to sidecar without TLS and with secured VPC peering.*

ELM-04	Localhost TLS Bypass		
Asset	`multichain-transporter: pkg/transporter/transporter.go`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The Sidecar RPC connection logic automatically disables TLS for localhost endpoints, making it impossible to establish secure TLS connections to localhost and increasing exposure to some security threats.

The problematic logic occurs at lines [**105-109**]:

```
if t.config.InsecureSidecarRPC || strings.Contains(t.config.SidecarRPCEndpoint, "localhost:") {
    creds = grpc.WithTransportCredentials(insecure.NewCredentials())
} else {
    creds = grpc.WithTransportCredentials(credentials.NewTLS(&tls.Config{InsecureSkipVerify: false}))
}
```

The `strings.Contains(t.config.SidecarRPCEndpoint, "localhost:")` condition forces insecure connections for any localhost endpoint, even when TLS is desired and available. This creates a security weakness by:

1. **Preventing defense-in-depth**: Eliminates TLS protection against unprivileged, malicious code running on the same host
2. **Reducing security posture**: Forces unencrypted communication in scenarios where TLS could provide additional protection
3. **Removing user choice**: Makes secure localhost connections impossible regardless of configuration intent, and may be unexpected.

Recommendations

Remove the automatic localhost exception and require explicit configuration:

```
if t.config.InsecureSidecarRPC {
    creds = grpc.WithTransportCredentials(insecure.NewCredentials())
} else {
    creds = grpc.WithTransportCredentials(credentials.NewTLS(&tls.Config{InsecureSkipVerify: false}))
}
```

This change would:

- Allow TLS connections to localhost when `InsecureSidecarRPC` is `false` (assuming a self-signed certificate for localhost has been installed on the system).
- Maintain explicit control through the `InsecureSidecarRPC` configuration flag.
- Provide defense-in-depth protection against same-host threats.

For additional hardening, consider implementing support for custom certificate configuration at the application-level to enable proper certificate validation for localhost and internal network scenarios.

Consider also logging a warning when unencrypted communication is used.

Resolution

The development team have acknowledged the issue, with respect to ELM-03, the TLS connection is not required.

ELM-05	`BlockNumber` Config Field and CLI Argument Has No Functional Purpose		
Asset	`multichain-transporter: pkg/transporter/transporter.go, cmd/transporter/main.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `BlockNumber` config field defined at `transporter.go:38` serves no functional purpose in the current implementation. It is only used for logging in two methods and does not affect any business logic or operational behaviour.

```
type Config struct {
    SidecarRPCEndpoint string
    CrossChainRegistry string
    DryRun             bool
    BlockNumber        uint64  // @audit Only used for logging
    SkipAVSTables      bool
    InsecureSidecarRPC bool
    L1ChainId          uint64
}
```

The field is populated from the CLI argument `--block-number` defined at `main.go:113` but only appears in logging statements at:

- `transporter.go:430` in `refreshGlobalTableRoot()`
- `transporter.go:443` in `Calculate()`

Both methods are unused as the main transport logic operates through block streaming from the sidecar RPC, not specific block number calculations. The actual block numbers used for transport operations come from the finalized block retrieved in `setupTransportComponents()`.

The impact depends on the originally intended functionality. If this field was meant to enable catching up on missed events after downtime by processing from a specific block number, its current non-functional state could be problematic for recovery scenarios.

Recommendations

Evaluate what the intended functionality for the `BlockNumber` field was supposed to be:

1. **If intended for recovery/catch-up functionality**: Implement proper block number handling in the streaming and transport logic to allow processing from a specified starting block.
2. **If no longer needed**: Remove the field and associated CLI flag entirely.

Resolution

The issue has been resolved in PR #19 by removing the unused `BlockNumber` field and CLI argument.

ELM-06	Fragile Logic In `shouldTriggerDailyTransport()`		
Asset	`multichain-transporter: pkg/transporter/transporter.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

`shouldTriggerDailyTransport()` contains fragile logic that fails to correctly identify some edge cases that involve blocks crossing the 14:00 UTC boundary. The implementation at lines [**247-251**] has two main flaws:

```
if prevTime.UTC().Format(time.DateOnly) != currTime.UTC().Format(time.DateOnly) {
    return false
}

return currTime.UTC().Hour() == 14 && prevTime.UTC().Hour() < 14
```

1. **Cross-day failure**: The day-equality check immediately returns `false` for blocks on different days, preventing legitimate cross-day 14:00 UTC boundary crossings from triggering daily transport.
2. **Exact hour requirement**: The final condition requires the current hour to be exactly 14, not at *or after* 14:00 UTC, causing failures when the next block arrives later than the 14:00-14:59 window.

Test cases provided alongside this report confirm these flaws. The business logic requires daily transport when crossing 14:00 UTC regardless of day boundaries, but the current implementation prevents this.

As the function is currently used, these faulty scenarios would not occur under normal network conditions where blocks are emitted multiple times per minute. Flaws would become apparent only in extreme network conditions with many skipped slots or if the `shouldTriggerDailyTransport()` logic is used in a different context.

Recommendations

The testing team recommends the following:

1. Evaluate whether the current logic is intended. If so, document these expectations clearly.
2. Consider refactoring the logic to more robustly handle described edge cases.
3. Incorporate further test cases.
4. As a matter of process, be careful when using LLM tools to generate both code and test cases. The generated test cases can be less reliable in confirming the correctness of the code. Use human reasoning to identify edge cases that the test cases miss.

Resolution

The issue has been resolved in PR #19 by refactoring the logic in `shouldTriggerDailyTransport()` to correctly handle edge cases involving the 14:00 UTC boundary.

ELM-07	Use Of `Fatalf()` Logging
Asset	`multichain-transporter: pkg/transporter/transporter.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `t.logger.Sugar().Fatalf()` logging mechanism is used in multiple places to log errors. However, `Fatalf()` calls `os.Exit(1)` which will result in the termination of the current process without any possibility of a graceful shutdown.

The errors being logged are somewhat deep into the application call-stack and appear unnecessary. The fatal logs are interspersed apparently arbitrarily amongst returned error values and it is not clear why some errors are fine and others should cause immediate halt. Immediate process termination is unnecessary.

Implementing immediate halts throughout the program can also make the control flow more difficult to reason about.

This also makes automated unit testing of that code infeasible.

Recommendations

Consider returning the error instead, and have the process retry the failed operation where appropriate.

If there are appropriate reasons to exit immediately, ensure these are documented clearly in the comments or error string.

In general, if critical errors require immediate shutdown, prefer wrapping them in an appropriate error type (e.g. named `TransportCriticalError`) and handling that specifically. Or consider using `Panicf()` (which raises a panic) and allows for unit testing with `recover()`.

Resolution

The issue has been resolved in PR #19 by replacing `Fatalf()` calls with appropriate error handling and logging.

ELM-08	Direct Error Comparisons Should Use `errors.Is()` For Wrapped Error Detection
Asset	`multichain-transporter: pkg/transporter/transporter.go, cmd/transporter/main.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The codebase contains direct error comparisons using `==` instead of `errors.Is()`, which can fail to detect errors when they are wrapped by higher-level error handling code. This pattern is flagged by the `errorlint` linter and can lead to incorrect program behavior.

Two relevant instances have been identified:

1. **Stream Termination Detection** — In `transporter.go:147`:

   ```
   if err == io.EOF {
       t.logger.Info("Server finished sending blocks")
       close(t.blockChan)
       <-t.processingDone
       return nil
   }
   ```

   Here gRPC's `ClientStream.RecvMsg()` is contractually required to return `io.EOF` directly (not wrapped) according to the official specification, `errors.Is()` is a safe rule-of-thumb.

2. **Context Cancellation Handling** — In `cmd/transporter/main.go:262`:

   ```
   if err == context.Canceled {
       l.Info("Transporter shutdown completed")
       return nil
   }
   ```

   This comparison *will fail* because any `context.Canceled` error would get wrapped in `t.Start()` before reaching this check:

   ```
   if err := t.subscribeToBlocks(ctx); err != nil {
       return fmt.Errorf("failed to subscribe to blocks: %w", err)
   }
   ```

   This results in graceful shutdowns appearing as errors instead of normal completion.

No associated security-related risk was identified in these instances. The impact could affect operational monitoring — where normal application shutdowns via SIGINT/SIGTERM are incorrectly reported as failures.

Recommendations

Replace direct error comparisons with `errors.Is()` calls to properly handle both direct and wrapped errors.

Consider including `errorlint` in the regular CI linting pipeline to automatically detect these patterns.

Resolution

The issues have been resolved in PR #19 by replacing direct error comparisons with `errors.Is()` calls.

ELM-09	Key Access Functions Return Zero Values Without Documentation
Asset	`eigenlayer-contracts: KeyRegistrar.sol, IKeyRegistrar.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

For `KeyRegistrar.sol` and its interface `IKeyRegistrar.sol`, the key access functions `getECDSAKey()`, `getECDSAAddress()`, and `getBN254Key()` return zero values for unregistered operators but this behaviour is not documented.

The interface documentation only mentions that functions revert if the OperatorSet uses the wrong curve type.

While `IOperatorTableCalculator` implementations developed by EigenLabs correctly call `isRegistered()`, this may not be obvious to AVS developers implementing custom calculator contracts, who may misinterpret zero values as valid keys.

Recommendations

Update the interface documentation in `IKeyRegistrar.sol` to specify that these functions return zero values for unregistered operators. Consider adding a note recommending callers use `isRegistered()` to verify registration status.

Resolution

The issue has been resolved in PR #1592 by updating the interface documentation to specify that these functions return zero values for unregistered operators.

ELM-10	`ChainManager` Not Thread-Safe Despite Concurrent Usage Patterns
Asset	`multichain-go: pkg/chainManager/chainManager.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `ChainManager` struct is not safe to use with multiple `goroutines` at the same time. It uses a Go map without locks, which can cause race conditions. Right now, this does not cause problems because the code only adds chains during startup before any reading happens.

The `ChainManager` stores chain connections in a simple Go map:

```go
type ChainManager struct {
    Chains map[uint64]*Chain
}
```

The `AddChain()` method writes to this map. The `GetChainForId()` method reads from this map. Go maps are not safe to write to during concurrent use:

```go
func (cm *ChainManager) AddChain(cfg *ChainConfig) error {
    // ...
    // @audit: Writing to map without locks
    cm.Chains[cfg.ChainID] = &Chain{
        config:    cfg,
        RPCClient: client,
    }
    return nil
}

func (cm *ChainManager) GetChainForId(chainId uint64) (*Chain, error) {
    // @audit: Reading from map without locks
    chain, exists := cm.Chains[chainId]
    fmt.Printf("Chain: %+v - exists: %+v - chainId %+v\n", chain, exists, chainId)
    // ...
}
```

The current code works fine because it only calls `AddChain()` during setup. After setup is done, only `GetChainForId()` gets called. But the code does not document this rule, and the methods are publicly exported for use in other packages.

Future developers might not know about this requirement. It is not unreasonable that later iterations include concurrent modifications like adding new chains to an already running transporter application.

Recommendations

Add a comment to the `ChainManager` struct to warn developers about thread safety. Also add a test to catch problems if someone changes how the code works later.

Ensure appropriate synchronisation is done if the ChainManager is modified during concurrent use.

Resolution

The issue has been resolved in PR #21 by adding appropriate synchronisation mechanisms to the `ChainManager` struct.

ELM-11	Incomplete Validation Of `config` Parameter In `_setOperatorSetConfig()`
Asset	`eigenlayer-contracts: CrossChainRegistry.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `_setOperatorSetConfig()` function is defined as follows:

```
function _setOperatorSetConfig(OperatorSet memory operatorSet, OperatorSetConfig memory config) internal {
        require(
            config.maxStalenessPeriod == 0 || config.maxStalenessPeriod >= _tableUpdateCadence, InvalidStalenessPeriod()
        );
        _operatorSetConfigs[operatorSet.key()] = config;
        emit OperatorSetConfigSet(operatorSet, config);
}
```

Here, the `config.maxStalenessPeriod` field is properly validated, however, the `config.owner` field is never validated to not be `address(0)`, or that the owner address is controlled by the AVS.

Recommendations

Evaluate whether `address(0)` is intended as a valid value for an OperatorSet owner (e.g. to signal that owner functionality is disabled).

Consider adding a check to ensure that `config.owner != address(0)`.

If stronger validation guarantees are important, one could implement AVS proof of possession checks for the owner address, similar to how it is done in the KeyRegistrar.

Resolution

The development team have acknowledged the issue with the intention that the address may be set to address(0) to indicate no owner.

ELM-12	Miscellaneous General Comments
Asset	*
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Unused Or Dead Code**

   Related Asset(s): multichain-transporter: pkg/transporter/transporter.go

   The following unused code was identified:

   (a) Two methods in the `Transporter` struct are unused dead code:

   - `(*Transporter).Calculate()` at `transporter.go:441-452`
   - `(*Transporter).refreshGlobalTableRoot()` at `transporter.go:428-439`

   The `refreshGlobalTableRoot()` method is only called from within `Calculate()`, but `Calculate()` itself is never invoked. Both methods contain only logging statements and return `nil` without performing actual functionality.

   (b) The `lastTransportDay` field in the `Transporter` struct is updated at `transporter.go:211` but is otherwise unused.

   These unused methods and fields have no identified security impact.

   Evaluate whether this is intentional and consider removing unused methods to clean up the codebase.

2. **Identified Typographical and Documentation Errors**

   In `eigenlayer-contracts`:

   (a) At `src/contracts/interfaces/IOperatorTableCalculator.sol:65` the doc comment contains a typo and broken reference:

   ```
   /// @dev This interface is implemented by the AVS in their own `OperatorTableCalculator` contract, see the
       ↪  Lay-Labs/middleware repository for an example implementation
   ```

   The repository name should be `Layr-Labs/eigenlayer-middleware`

3. **Redundant Check For `referenceTimestamp` In `updateOperatorTable()`**

   Related Asset(s): OperatorTableUpdater.sol, BN254CertificateVerifier.sol, ECDSACertificateVerifier.sol

   The `updateOperatorTable()` function has the following check:

   ```
   require(
           referenceTimestamp
               > IBaseCertificateVerifier(getCertificateVerifier(curveType)).latestReferenceTimestamp(operatorSet),
           TableUpdateForPastTimestamp()
       );
   ```

   However, this check is already being performed within both `BN254CertificateVerifier.updateOperatorTable()` ([**70**]) and `ECDSACertificateVerifier.updateOperatorTable()` ([**56**]) making it redundant.

   Consider removing the redundant code.

4. **Lack Of Zero-Address Validation**

   Related Asset(s): eigenlayer-contracts: CrossChainRegistry.sol, CrossChainRegistryStorage.sol, ECDSACertificateVerifierStorage.sol, BN254CertificateVerifierStorage.sol, OperatorTableUpdaterStorage.sol

   Several functions and constructors are missing validation that address parameters are not the zero-address (`address(0)`). This can help protect against common misconfiguration where deployment methods proceed when constructor parameters are accidentally omitted.

   The following instances were identified:

   (a) **Function Parameter Validation**:
   - `addChainIDsToWhitelist()`: The `operatorTableUpdaters[i]` parameters are never validated to not be `address(0)`, despite `chainID` validation being present.
   - `setOperatorTableCalculator()`: The `operatorTableCalculator` parameter is never validated to ensure it is not `address(0)`, while the `operatorSet` parameter is validated via the `isValidOperatorSet()` modifier.

   (b) **Constructor Parameter Validation**:
   - `CrossChainRegistryStorage.sol`: the `allocationManager` and `keyRegistrar` parameters.
   - `OperatorTableUpdaterStorage.sol`: the `bn254CertificateVerifier` and `ecdsaCertificateVerifier` parameters.
   - `BN254CertificateVerifierStorage.sol`: the `operatorTableUpdater` parameter.
   - `ECDSACertificateVerifierStorage.sol`: the `operatorTableUpdater` parameter.

   Consider adding zero-address checks in the identified functions and constructors to verify that address parameters are not `address(0)`.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team's responses to the miscellaneous issues are as follows:

1. Fixed in PR #19
2. Fixed in PR #1592
3. Acknowledged
4. Acknowledged

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework and go tests were used and the output is given below.

```
$ go test -v -run ^TestShouldTriggerDailyTransport$ github.com/Layr-Labs/multichain-transporter/pkg/transporter
=== RUN   TestShouldTriggerDailyTransport
=== RUN   TestShouldTriggerDailyTransport/blocks_on_separate_days_crossing_14:00_UTC
    transporter_test.go:90: Test blocks on separate days crossing 14:00 UTC failed: expected true, got false. should trigger when
          ↪  crossing with an extra day
=== RUN   TestShouldTriggerDailyTransport/blocks_on_separate_days_crossing_14:00_UTC_but_times_are_later
    transporter_test.go:90: Test blocks on separate days crossing 14:00 UTC but times are later failed: expected true, got false.
          ↪  should trigger when crossing with an extra day
=== RUN   TestShouldTriggerDailyTransport/blocks_on_separate_days_but_not_crossing_14:00
=== RUN   TestShouldTriggerDailyTransport/crossing_14:00_more_than_an_hour_later
    transporter_test.go:90: Test crossing 14:00 more than an hour later failed: expected true, got false. should trigger when
          ↪  crossing 14:00 but more than an hour has elapsed
--- FAIL: TestShouldTriggerDailyTransport (0.00s)
    --- FAIL: TestShouldTriggerDailyTransport/blocks_on_separate_days_crossing_14:00_UTC (0.00s)
    --- FAIL: TestShouldTriggerDailyTransport/blocks_on_separate_days_crossing_14:00_UTC_but_times_are_later (0.00s)
    --- PASS: TestShouldTriggerDailyTransport/blocks_on_separate_days_but_not_crossing_14:00 (0.00s)
    --- FAIL: TestShouldTriggerDailyTransport/crossing_14:00_more_than_an_hour_later (0.00s)
FAIL
FAIL    github.com/Layr-Labs/multichain-transporter/pkg/transporter     0.042s
FAIL
```

```
$ go test -race ./multichain/chainManager_test.go
Chain: <nil> - exists: false - chainId 0
Chain: <nil> - exists: false - chainId 1
Chain: <nil> - exists: false - chainId 2
 ...
==================
WARNING: DATA RACE
Write at 0x00c0002e0360 by goroutine 9:
  runtime.mapassign()
      <gopath>/src/internal/runtime/maps/runtime_swiss.go:191 +0x0
  github.com/Layr-Labs/multichain-go/pkg/chainManager.(*ChainManager).AddChain()
      <redacted>/multichain-go/pkg/chainManager/chainManager.go:79 +0x2c4
  command-line-arguments.TestChainManagerRacePOC.func1()
      <project_root>/multichain/chainManager_test.go:19 +0x46

Previous read at 0x00c0002e0360 by goroutine 10:
  runtime.mapaccess1()
      <gopath>/src/internal/runtime/maps/runtime_swiss.go:43 +0x0
  github.com/Layr-Labs/multichain-go/pkg/chainManager.(*ChainManager).GetChainForId()
      <redacted>/multichain-go/pkg/chainManager/chainManager.go:96 +0x64
  command-line-arguments.TestChainManagerRacePOC.func2()
      <project_root>/multichain/chainManager_test.go:29 +0x64

Goroutine 9 (running) created at:
  command-line-arguments.TestChainManagerRacePOC()
      <project_root>/multichain/chainManager_test.go:17 +0x152
  testing.tRunner()
      <gopath>/src/testing/testing.go:1792 +0x225
  testing.(*T).Run.gowrap1()
      <gopath>/src/testing/testing.go:1851 +0x44

Goroutine 10 (running) created at:
  command-line-arguments.TestChainManagerRacePOC()
      <project_root>/multichain/chainManager_test.go:27 +0x1fc
  testing.tRunner()
      <gopath>/src/testing/testing.go:1792 +0x225
  testing.(*T).Run.gowrap1()
      <gopath>/src/testing/testing.go:1851 +0x44
==================
Chain: <nil> - exists: false - chainId 3
Chain: <nil> - exists: false - chainId 4
Chain: <nil> - exists: false - chainId 5
```

```
Chain: <nil> - exists: false - chainId 6
...
Chain: &{config:0xc00020a000 RPCClient:0xc000218008} - exists: true - chainId 0
Chain: &{config:0xc00020a030 RPCClient:0xc000218010} - exists: true - chainId 1
...
--- FAIL: TestChainManagerRacePOC (0.02s)
    testing.go:1490: race detected during execution of test
FAIL
FAIL  command-line-arguments  0.034s
FAIL
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

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

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].