controlplane

CLOUD NATIVE COMPUTING FOUNDATION



LINKERD

Linkerd Threat Model and End-User Hardening Guide

November 2026

Contents

Contents **2**
Executive Summary **4**
Introduction **6**
Scope **6**
In scope 6
Out of scope 7
Threat Landscape **9**
CIA impact assessment 9
Relevant threat actors 11
In scope 11
Out of scope 13
Example Architectures **15**
Single-cluster architecture overview 16
Multi-tenancy 16
Linkerd configuration 16
Data flow diagram 16
Hierarchical Multi-Cluster Architecture Overview **18**
Clusters and trust boundaries 18
Networking 18
Multi-tenancy 19
Identity federation / Chain of trust 21
Authorisation policies 22
Data flow diagram 22
Flat network multi-cluster architecture overview **24**
Clusters and trust boundaries 24
Data flow diagram 24
Data **26**
Data dictionary 26
Priority legend 26
Key Threats and Recommendations **29**
Severity 29
Categories 29
Threats 30
T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys 31
T-02 - Unauthorised access to meshed services due to permissive default inbound policy 33
T-03 - Unmeshed workload admission due to permissive proxy injector webhook failure

policy 35
T-04 - Overly permissive or absent authorisation policies 37
T-05 - Unauthorised modification of Linkerd policy resources 39
T-06 - Identity spoofing via Kubernetes Service Account token theft or RBAC abuse 41
T-07 - Ingress mode proxy open relay via spoofed l5d-dst-override header 42
T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation 45
T-09 - Linkerd control plane compromise 48
T-10 - Identity spoofing or denial of service via trust bundle tampering 50
T-11 - Cross-cluster identity obfuscation via gateway confused deputy 52
T-12 - Cross-cluster lateral movement in flat network multicluster mode 54
T-13 - Remote cluster Kubernetes API access via link credentials theft 56
T-14 - Data exposure and compliance violations via inappropriate service mirroring or federation 58
T-15 - Cross-cluster policy drift 60
T-16 - Cluster compromise via Linkerd-Viz bundled Prometheus nodes/proxy GET permissions 62
T-17 - Mesh-wide outage via trust anchor or identity issuer certificate expiration 64
T-18 - Linkerd control plane denial of service 66
T-19 - Mesh management outage via webhook TLS certificate expiration 68
T-20 - Cross-cluster traffic disruption via out-of-sync trust anchor rotation 70
T-21 - Multicluster gateway overexposure and denial of service 71
T-22 - Residual access via incomplete multicluster unlinking 73
T-23 - Workload identity spoofing via proxy credential theft 74
T-24 - L7 policy evasion via protocol detection bypass or opaque port misconfiguration 76
T-25 - L7 policy evasion via proxy-application parser differentials 78
T-26 - Overly permissive or absent EgressNetwork policies 80
T-27 - Plaintext data interception via local loopback snooping 82
T-28 - Linkerd-Viz Tap API abuse for real-time traffic interception 84
T-29 - Denial of service and infrastructure overwhelm via telemetry resource exhaustion (CVE-2025-43915) 86
T-30 - Pod denial of service via proxy shutdown endpoint 88
Attack trees **90**
Attack tree node legend 90
Mesh PKI compromise 91
Compromised workload - confidentiality threats 92
Multicluster threats 93
Hardening **94**
Priority 94
Recommendations 95
R-01: Store the Trust Anchor Private Key Externally 95
R-02: Default Inbound Policy Hardening 95
R-03: Explicitly Define Least Privilege Authorisation Policy Resources 96

R-04: Proxy Injector Webhook Failure Policy Hardening 96
R-05: Restrict Linkerd Data Plane Annotations via Admission Controller 96
R-06: Least Privilege RBAC to Linkerd Namespaces 97
R-07: Restrict and Audit Access to Secrets 97
R-08: Least Privilege Access to Linkerd Policy Resources 98
R-09: Encrypt and Protect Kubernetes Secrets at Rest 98
R-10: Enforce Default-Deny Network Policies 98
R-11: Workload Runtime Security Hardening 98
R-12: Centralise Policy Management via GitOps 99
R-13: Certificate Lifecycle Management and Automation 100
R-14: Implement Multi-Cluster Network Policies in Flat Network Architectures 100
R-15: Least Privileged Multicluster Link Credentials 100
R-16: Restrict Service Mirroring and Federation via Admission Controller 101
R-17: Restrict Multicluster Gateway Access Controls and Network Exposure 101
R-18: Configure Ingress Controller to Strip l5d-dst-override Header 102
R-19: Remove Excessive Prometheus Privileges 103
R-20: Deploy Linkerd Control Plane in High Availability Mode 103
R-21: Enforce Pod Security Standards (PSS) 103
R-22: Enable the Linkerd CNI Plugin 104
R-23: Opt Out of Service Account Token Automounting 104
R-24: Enforce ExternalNetwork Policies for Egress Traffic 105
R-25: Restrict Linkerd-Viz Tap API Access 105
R-26: Exclude Sensitive Headers from Being Captured by Linkerd-Viz Tap 105
R-27: Explicitly Declare Application Protocols for Linkerd Server Resources and Kubernetes Services 105
R-28: Configure Robust Logging and Monitoring 106
R-29: Implement Application Level Authorisation for Sensitive Services 107
R-30: Deploy a Runtime Security Solution 107
R-31: Configure Short-Lived Workload Certificates 108
R-32: Ensure the Proxy Shutdown Endpoint is Disabled 108
R-33: Ensure Hostname Labels for Metrics are Disabled 108

Appendix **109**
Considerations for Buoyant Enterprise for Linkerd 109
References 111

About **112**

Team **112**

Reviewers **112**

Executive Summary

This guide presents a threat-model–driven approach to hardening the Linkerd service mesh. It serves as a practical, production-ready blueprint for securing Kubernetes deployments with the Linkerd service mesh, going beyond traditional penetration testing to address critical security considerations. Particular focus is given to cross-cluster communications and other high-risk areas relevant to highly regulated industries, including financial services.

Linkerd operates as a service mesh implemented via a per-cluster control plane and distributed data plane proxy sidecars. It intercepts and manages pod-to-pod traffic to enforce mTLS, dynamic routing rules, and fine-grained authorisation policies for meshed workloads. As a result, misconfigurations in Linkerd's cryptographic materials, authorisation policy definitions, webhook settings, or associated role-based access control (RBAC) can significantly weaken zero-trust policy enforcement or severely disrupt availability. This is particularly relevant in environments where Linkerd is relied upon as a primary control for zero-trust architectures and secure inter-service communication.

A threat modeling exercise identified a range of risks affecting Linkerd across both single-cluster and multi-cluster architectures. These risks span multiple domains, including cryptographic key management, admission control, Linkerd control and data plane resilience, authorisation policy enforcement, RBAC, cross-cluster boundary protection, network isolation, secret storage, certificate lifecycle management, and how the underlying Kubernetes platform's security posture can impact the integrity and security guarantees of the mesh.

The analysis is followed by hardening recommendations, which provide detailed practical guidance on mitigating the identified risks across Linkerd and supporting architectural components. For selected high-impact findings, "Attack Trees" are included to visually map the pathways an attacker could take to exploit combinations of misconfigurations and illustrate how the recommended security controls reduce overall risk.

This document is intended for platform engineers and security teams seeking to deploy Linkerd securely in production Kubernetes environments - including single or multi-cluster deployments. It complements Linkerd’s native security model by addressing risks that arise in real-world operational deployments.



—
Andrew Martin
CEO, ControlPlane

Introduction

Linkerd is a lightweight, production-ready service mesh for Kubernetes that provides secure, observable, and reliable service-to-service communication. It deploys sidecar proxies alongside application containers, automatically encrypting traffic with mutual TLS (mTLS) and enforcing identity-based authorisation between microservices.

Linkerd can also operate in multi-cluster environments, enabling services across separate Kubernetes clusters to communicate securely and transparently. In this setup, Linkerd establishes a connection between clusters that's reliable, encrypted, and authenticated on both sides with mTLS, enabling organisations to maintain zero-trust communication, observability, and reliability even in distributed, multi-cluster architectures.

Scope

The focus of this threat-model–driven approach to end-user hardening is defined within this section.

In scope

This report provides a threat-model-driven security reference for Linkerd end users focusing on practical hardening guidance for production Kubernetes environments, targeting Linkerd 2.20, which is the latest stable version at the time of writing.

The analysis targeted the following open-source components and functionalities of Linkerd:

- **Linkerd control plane:** The overarching management plane that dictates the routing behaviour, security posture, and identity provisioning of the entire Linkerd mesh
    - **Proxy injector:** A Kubernetes mutating admission webhook that intercepts pod creation events to automatically inject the proxy sidecar and init container configurations into workloads
    - **Identity service:** Acts as a Certificate Authority (CA) that receives Certificate Signing Requests (CSRs) from proxies and issues short-lived workload certificates to enable mTLS communications
    - **Destination service:** Used by data plane proxies to fetch information about service discovery, policies, and service profiles
- **Linkerd data plane:** Linkerd micro-proxies and setup containers running alongside application workloads in their respective pods
    - **Proxy:** An lightweight Rust micro-proxy that transparently handles mTLS encryption, Layer 7 routing, load balancing, and telemetry generation for its associated application container
    - **Linkerd init container:** A short-lived setup container that configures `iptables` rules within the pod's network namespace to redirect all inbound and outbound

TCP traffic through the proxy sidecar
- **Multi-cluster communication:** Linkerd's functionalities enabling mTLS-authenticated, service-to-service communication across cluster boundaries
    - **Hierarchical mode:** A multi-cluster architecture where cross-cluster traffic flows securely through a dedicated Linkerd gateway, only requiring the gateway IP to be reachable
    - **Flat network mode:** A multi-cluster architecture where pods communicate directly with remote pod IP addresses across clusters without traversing a gateway, requiring a flat network topology

The `linkerd-viz` extension provides an in-cluster observability stack and debugging features, including a web dashboard, a bundled Prometheus instance, and the `tap` tool which allows users to inspect traffic in real time. Due to the broad scope of the assessment, threats related to `linkerd-viz` were only partially covered.

Security considerations focused on deployment architecture, configuration, and operational practices that influence the security posture of Linkerd installations. In particular, we examine:

- Secret management and certificate storage
- Kubernetes RBAC considerations for Linkerd components and policy resources
- Network isolation and multi-cluster security considerations
- Container and workload security practices and their impact on the integrity of the mesh and its security guarantees

The goal is to identify common misconfigurations or systemic architectural risks and provide actionable recommendations for securely deploying and operating Linkerd in production environments.

Out of scope

This document does not constitute a vulnerability assessment, penetration test, or comprehensive security review of Linkerd or any of the associated components. The guidance provided should not be interpreted as an evaluation of the security posture of specific deployments.

The following areas are explicitly out of scope:

- Buoyant Enterprise for Linkerd
- Buoyant Cloud
- Linkerd CLI
- Non-Kubernetes workloads (mesh expansion)

A comprehensive threat model of Linkerd’s software supply chain is out of scope. However, downstream risks arising from a compromised workload are considered in scope and addressed where relevant.

Broader systemic threats, such as malicious and intentional source code injections by maintainers, are treated as inherent risks within the open source ecosystem and are partially mitigated by Linkerd's **CNCF Graduated status** (achieved in 2021), which mandates rigorous governance and community oversight.

Additionally, this guide does not address:

- Threats associated with advanced persistent threat (APT) actors, for example nation state actors exploiting 0-day vulnerabilities in Linkerd components
- Risks that apply only to unsupported or end-of-life versions of Linkerd

Threat Landscape

This section outlines the threat landscape in relation to Kubernetes environments utilising service mesh technologies and evaluates the associated operational impact. It combines a CIA-based impact assessment (used to consistently rate confidentiality, integrity, and availability risks across production and pre-production clusters), with an analysis of the relevant threat actors and their capabilities, motivations, and examples of common attacks. Together, these perspectives provide a structured basis for understanding how different adversaries can exploit service mesh workflows and cluster configurations, and how such events translate into real-world business and operational risk.

CIA impact assessment

This section defines a structured approach for correlating the identified severity of a risk with its operational impact, utilizing the CIA (Confidentiality, Integrity, and Availability) triad.

Contextual Factors

The assigned risk levels of a specific environment may be adjusted upward or downward based on data classification and the surrounding business context. For instance, the compromise of highly sensitive business data carries a significantly greater impact than the exposure of non-sensitive data. Similarly, a loss of availability in a production cluster poses a more severe impact than a similar compromise in a pre-production cluster.

Typical Risk Ratings

This table outlines a risk rating profile for service mesh-related security threats, categorised by CIA. Each rating reflects the potential impact on production and pre-production environments, helping to guide prioritisation and mitigation efforts.

Confidentiality	High	Total loss of cryptographic trust (e.g., via compromised trust anchor or issuer keys), resulting in loss of confidentiality over sensitive business data
	Medium	Attacker gains temporary time-bound access to limited business data Information which could be useful for attacker reconnaissance is leaked
	Low	Non-sensitive information leakage

Integrity	High	Collapse of cluster-wide zero-trust boundaries due to cryptographic trust subversion (e.g., rogue CA injection) Subversion of security controls via global mesh configuration tampering
	Medium	Localised tampering with specific application's policies causing isolated corruption
	Low	Non-critical configuration drift without leading to policy bypass Non-critical operations blocked due to misconfiguration
Availability	High	Mesh-wide DoS disrupting all intra-cluster or cross-cluster traffic in a production environment
	Medium	Specific meshed application is blocked for a significant period Disruption of intra-cluster or cross-cluster traffic in a pre-production environment
	Low	Temporary disruption of a non-critical application

Relevant threat actors

The following table details the specific types of threat actors relevant to the service mesh threat landscape alongside their typical capabilities, motivations, and examples of attacks.

In scope

Threat Actor	Capability	Personal Motivation	Sample Linkerd Attacks
Cluster Administrator (Malicious)	Full control over all cluster resources: including CRDs, RBAC, `Secrets`, admission webhooks, and Linkerd configuration	Financial gain, coercion, insider threat, or operational misuse	Extract Trust Anchor or Identity Issuer private keys from `Secrets` and use them to mint arbitrary issuer or workload certificates Tamper with the trust bundle to inject a malicious CA or delete the legitimate root of trust Disable or weaken security controls (such as the cluster-wide default inbound policy or webhook failure policy)
Cluster Administrator (Accidental)	Full control over all cluster resources: including CRDs, RBAC, `Secrets`, admission webhooks, and Linkerd configuration	Operational efficiency, human error	Accidentally misconfiguring Linkerd settings, e.g. leaving insecure defaults such as a permissive default inbound policy or using a self-signed trust anchor and storing the private key in-cluster Accidentally granting RBAC principals overly broad permissions to resources in the `linkerd` namespace
Namespace / Tenant Administrator (Malicious)	Administrative control within a specific tenant namespace, including permissions to manage `Server`, `AuthorizationPolicy`, `HTTPRoute`, `GRPCRoute`, `ExternalNetwork` and other namespaced Linkerd policy resources	Privilege escalation, lateral movement, tenant isolation bypass, financial gain	Disable sidecar proxy injection for the tenant namespace Misconfigure or delete authorisation policies to allow unauthorised data access or lateral movement Misconfigure or delete egress policies to enable data exfiltration or C2 communications

Namespace / Tenant Administrator (Accidental)	Administrative control within a specific tenant namespace, including permissions to manage `Server`, `AuthorizationPolicy`, `HTTPRoute`, `GRPCRoute`, `ExternalNetwork` and other namespaced Linkerd policy resources	Operational efficiency, human error	Misuse privileges to override the default inbound policy for the tenant namespace, setting it to a more permissive value, for reasons of operational expediency Accidentally misconfigure Linkerd authorisation or egress policies in the namespace
Internal Platform Engineer (Malicious)	Authenticated Kubernetes user with restricted permissions	Privilege escalation or unauthorised data access	Attempt to read `Secrets` containing TLS private keys Exploit overly permissive RBAC, for example to obtain a `ServiceAccount` token and impersonate its identity Abuse the Linkerd-Viz Tap API to intercept traffic
Internal Platform Engineer (Accidental)	Authenticated Kubernetes user with restricted permissions	Operational efficiency, human error	Accidentally misconfigure ingress controller, not stripping or overriding the `l5d-dst-override` header, allowing the Linkerd ingress-mode proxy to act as an open relay
Internal Developer / Workload Owner (Malicious)	Ability to deploy applications	Privilege escalation or financial gain	Deploy unmeshed workload (e.g., by explicitly disabling proxy injection via an annotation) Deploy workload with intentionally insecure proxy configuration (e.g., misconfigured skip ports, reduced log level, or overridden proxy sidecar image) Abuse lack of granular namespace segregation and assign arbitrary `ServiceAccount` to a workload to abuse its identity

Internal Developer / Workload Owner (Accidental)	Ability to deploy applications	Operational efficiency, human error	Deploy workload with excessive Linux capabilities that could enable proxy bypass in case of container compromise Accidentally deploy workload without Linkerd proxy, e.g. by running the workload with host networking Accidentally evade outbound proxy, e.g. by having an application container run as the same UID as the Linkerd proxy
External vandal (script kiddie, trespasser)	Exploit publicly exposed services using common tools and known vulnerabilities	Curiosity or notoriety through defacement and service disruption	Attempt SSRF-like attacks by spoofing the `l5d-dst-override` header in inbound requests to meshed ingress endpoints with ingress-mode proxy Perform denial-of-service attacks against the Linkerd gateway

Out of scope

Threat Actor	Capability	Personal Motivation	Sample Linkerd Attacks
Advanced Persistent Threat / Organized Crime Group	Sophisticated attacker with resources for targeted attacks and long-term persistence including the usage of custom exploits and coercion	Data theft, espionage, financial fraud or ransom	Compromise external root CA private keys to intercept and decrypt sensitive internal communications Leverage Linkerd misconfigurations or supply chain flaws to insert rogue trust anchors into the cluster Exploit 0-day vulnerabilities in the Linkerd proxy or control plane components Capturing, exfiltrating and storing encrypted traffic that does not use post-quantum safe TLS (PQ-TLS), enabling retroactive compromise of sensitive communications

Supply Chain Attackers (Linkerd maintainers and contributors) (Malicious)	Ability to introduce malicious code through open source contributions or compromised dependencies	Political sabotage or financial gain	Deliberately introduce malicious logic into Linkerd
Supply Chain Attackers (Linkerd maintainers and contributors) (Accidental)	Ability to introduce malicious code through open source contributions or compromised dependencies	Human error	Accidentally approve or merge insecure changes from contributors or compromised third party dependencies

Example Architectures

This section presents three representative reference architectures for Linkerd deployment, designed to frame the threat modelling and hardening guidance that follows.

1. The Single-cluster architecture is presented first as a simple and widely-applicable baseline
2. The Hierarchical multi-cluster architecture follows, utilising Linkerd's hierarchical (gateway) multi-cluster communication mode, representing a mature enterprise deployment pattern suitable for organisations with environmental separation or multi-region operational requirements
3. The Flat network multi-cluster architecture follows, which is architecturally similar to the previous example but utilises Linkerd's flat network (peer-to-peer) multicluster communication mode and uses federated service to load-balance cross-cluster traffic between multiple management clusters

All three architectures incorporate intentional misconfigurations designed to reflect realistic operational decisions rather than obviously insecure configurations. Neither should be treated as a production blueprint. Significant flaws are included throughout to motivate the subsequent threat model and following recommendations.

Single-cluster architecture overview

This architecture represents a single cluster Linkerd service mesh deployment which typically reflects the experience of practitioners who have installed Linkerd via the default Helm chart and are working toward a hardened posture.

Multi-tenancy

A single Kubernetes cluster hosting application workloads and the Linkerd control plane. This type of deployment typically reflects the experience of practitioners who have installed Linkerd via the default Helm chart and are working toward a hardened posture.

Namespaced `AuthorizationPolicy` and `EgressNetwork` resources, authored and managed by the platform operations team, are used to restrict ingress traffic to namespaced `Server` and `HTTPRoute` resources managed by application teams and egress traffic originating from application teams' namespace respectively.

Linkerd configuration

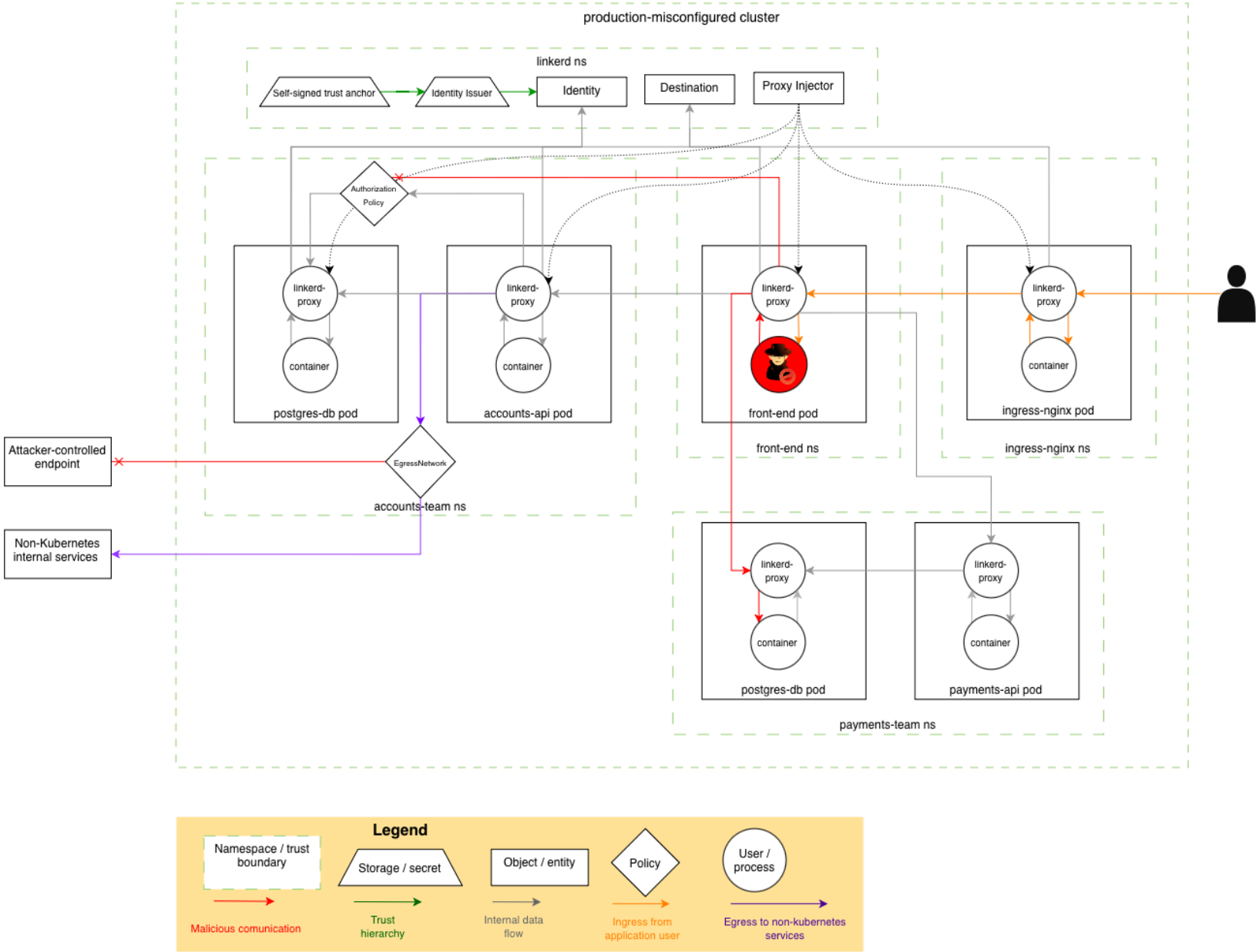
The trust anchor is a self-signed certificate with the private key stored in-cluster. An identity issuer certificate with 1 year validity is generated during installation and is not automatically rotated.

Linkerd is deployed with many of the default configuration values from the default Helm chart, for example:

- `proxy.defaultInboundPolicy: "all-unauthenticated"`: The default inbound policy when no `Server` selects a pod allows all traffic
- `webhookFailurePolicy: Ignore`: If the Linkerd proxy injector webhook is unavailable or encounters errors during pod creation, the proxy injection process is silently skipped, and the application pod is scheduled without the Linkerd sidecar injected
- `cniEnabled: false`: The Linkerd CNI is not used, so a `NET_ADMIN` capability is required by the `proxy-init` container when injecting the Linkerd proxy
- `controllerReplicas: 1`: A single replica is deployed for each Linkerd control plane component pod

Data flow diagram

The Data Flow Diagram (DFD) below describes the flow of data between the various processes, entities and data stores in the system, as well as the trust boundaries between components.


production-misconfigured cluster
linkerd ns
Self-signed trust anchor
Identity Issuer
Identity
Destination
Proxy Injector
Authorization Policy
linkerd-proxy
container
postgres-db pod
accounts-api pod
front-end pod
ingress-nginx pod
front-end ns
ingress-nginx ns
Attacker-controlled endpoint
EgressNetwork
accounts-team ns
Non-Kubernetes internal services
payments-api pod
payments-team ns
Legend
Namespace / trust boundary
Storage / secret
Object / entity
Policy
User / process
Malicious comunication
Trust hierarchy
Internal data flow
Ingress from application user
Egress to non-kubernetes services

Hierarchical Multi-Cluster Architecture Overview

This section describes a multi-cluster Linkerd service mesh deployment implemented using the Linkerd gateway in a hierarchical Hub-and-Spoke topology. This infrastructure is intended to be representative of a geographically distributed, regulated financial services organisation, in which:

- A global platform engineering function (the Hub) maintains control over shared services
- Regional application domains (the Spokes) operate semi-autonomously, hosting jurisdiction-specific workloads while utilising shared services from the Hub

This multi-cluster model enables segregation of duties and jurisdictional isolation, consistent with regulatory expectations for defense-in-depth and blast-radius containment.

Clusters and trust boundaries

Two Kubernetes clusters represent two distinct trust zones:

- `mgmt-hub`: Represents a highly restricted administrative security zone. This cluster provides centrally governed platform services and data consumed by regional clusters. It is accessed exclusively by platform SRE teams
- `regional-spoke-misconfigured`: Represents an organisation's deployments within a specific region. This cluster operates an independent Kubernetes control plane, hosts application workloads from various isolated tenants and is accessed by both internal application developers and external application users. It consumes shared services and data from the `mgmt-hub` cluster and uses an intentionally flawed Linkerd configuration to demonstrate the impact of an insecure service mesh

Aside from the Linkerd-specific issues affecting the `regional-spoke-misconfigured` cluster, the remainder of the entire multi-cluster infrastructure is considered to be hardened in alignment with security baselines, such as NIST 800-53 rev 5.

Networking

Clusters are isolated from a networking perspective, with only the Gateway IP needing to be reachable from spoke cluster application workloads. However, the Linkerd control plane in the spoke cluster requires network reachability to the hub cluster's Kubernetes API to facilitate service mirroring.

- **Ingress Control:** The `mgmt-hub` cluster can not be reached from the external internet. All external traffic to the `regional-spoke` cluster is funneled through a Linkerd-injected Ingress Controller. This ensures that mTLS is enforced on "north-south" traffic as soon as it enters the cluster

- **Cluster Isolation:** Clusters reside in separate non-routable, isolated subnets, mimicking a real-world multi-region or hybrid cloud. Inter-cluster traffic is routed through the Linkerd gateway
- **Egress:** Cluster workloads are only permitted restricted outbound connectivity to approved non-Kubernetes resources, such as other internal services, via the Linkerd `EgressNetwork` resource

Multi-tenancy

Namespace isolation is used to enforce soft multi-tenancy across both clusters, ensuring that the compromise of one team's application does not grant immediate access to resources owned by another team.

The following table describes the namespaces and their corresponding tenants within the `mgmt-hub` cluster:

`mgmt-hub` **Namespace(s)**	**Tenants / Responsible Team(s)**	**Purpose**
`kube-system`	Platform Engineering	Hosts core Kubernetes control plane components, including the API server and scheduler. Forms the foundation of all management operations
`linkerd`	Platform Engineering Security Operations	Contains management cluster's Linkerd control plane components, handling sidecar configuration and identity services
`linkerd-multicluster`	Platform Engineering	Operates the Linkerd gateway receiving encrypted traffic from spoke clusters
`cert-manager`	Security Operations, PKI	Automates identity issuer and webhook certificates rotation

`linkerd-viz`	Platform Engineering Internal Audit	Per-cluster operational observability extension, enabling immediate traffic analysis and troubleshooting
`shared-platform-services`	Platform Engineering Administration	Contains any shared services and data consumed by workloads in regional spoke clusters

Similarly, the following table describes the namespaces and their corresponding tenants within the `regional-spoke` cluster:

`regional-spoke` **Namespace(s)**	**Tenants / Responsible Team(s)**	**Purpose**
`kube-system`	Platform Engineering	Hosts core Kubernetes control plane components, including the API server and scheduler. Forms the foundation of all management operations
`linkerd`	Platform Engineering Security Operations	Coordinates the `regional-spoke` Linkerd control plane, handling sidecar configuration and identity services across the management cluster
`linkerd-multicluster`	Platform Engineering	Operates the service mirror controller that watches the remote cluster for exported services and mirrors them locally
`ingress-nginx`	Platform Engineering	Handles initial termination of external user requests and applies edge-level traffic shaping before passing requests into the mesh

`front-end`	Development	Hosts user-facing web services; acts as the primary entry point for business logic and is the first line of defense within the application tier
`back-end`	Development	Contains internal APIs and sensitive data processing, restricted to internal mesh traffic via identity-based authorisation policies

Identity federation / Chain of trust

For Linkerd's native multi-cluster extension to function, both clusters must verify each other's certificates against a shared trust anchor certificate. The following four-tier Public Key Infrastructure (PKI) hierarchy is used to minimise the blast radius of a key compromise:

- **Tier 1: Corporate Root CA (Offline/Highly Secure):** The root of trust for the whole organisation. To prevent global compromise, the private key is never stored in Kubernetes; it resides in an external HSM or Vault. Its sole purpose is to sign Tier 2 Intermediate CAs, ensuring the root CA remains air-gapped from all cluster runtimes
- **Tier 2: Intermediate CA (The Trust Anchor):** Represented as the `identityTrustAnchorsPEM` in Linkerd. This is the **Public Certificate** shared by both clusters. The trust anchor private key is stored outside of Kubernetes
- **Tier 3: Identity Issuer (Cluster-Specific):** Each cluster has its own Identity Issuer used to sign local workload certificates. In the Hardened cluster, `cert-manager` is used to automatically rotate the Identity Issuers with a short TTL (set to 90 days in this example). In the `regional-spoke-misconfigured`, this is a long-lived, static Kubernetes Secret
- **Tier 4: Workload/Leaf Certificates:** Short-lived certificates issued to every Linkerd sidecar proxy. This example deployment uses the Linkerd default 24 hours TTL, though the duration can be modified as needed

This PKI hierarchy bounds the impact of a key compromise in the following ways:

- The higher tiers (1 and 2) are primarily protected through offline or external key custody and controlled rotation
    - The Tier 1 Corporate Root CA private key never enters the Kubernetes environment and the private key for the Tier 2 Trust Anchor is stored externally

- Short-lived issuer and workload certificates limit the useful lifetime of compromised keys at Tiers 3 and 4
    - Once a compromised issuer certificate expires, certificates chaining through it are no longer trusted. Continuing to mint trusted workload identities would require an attacker to obtain a new issuer signed by the trust anchor
    - Automatic certificate rotation using `cert-manager` removes the requirement for manual intervention in the `mgmt-hub` cluster for rotating the Identity Issuer. This mitigates the risk of expired Identity Issuer credentials and ensures that even if a private key is exfiltrated, its window of utility is limited
    - Dedicated Identity Issuers per cluster isolate the remediation effort. While a compromised issuer can mint certificates trusted by both clusters, having separate issuers means rotation is only required in the affected cluster

Authorisation policies

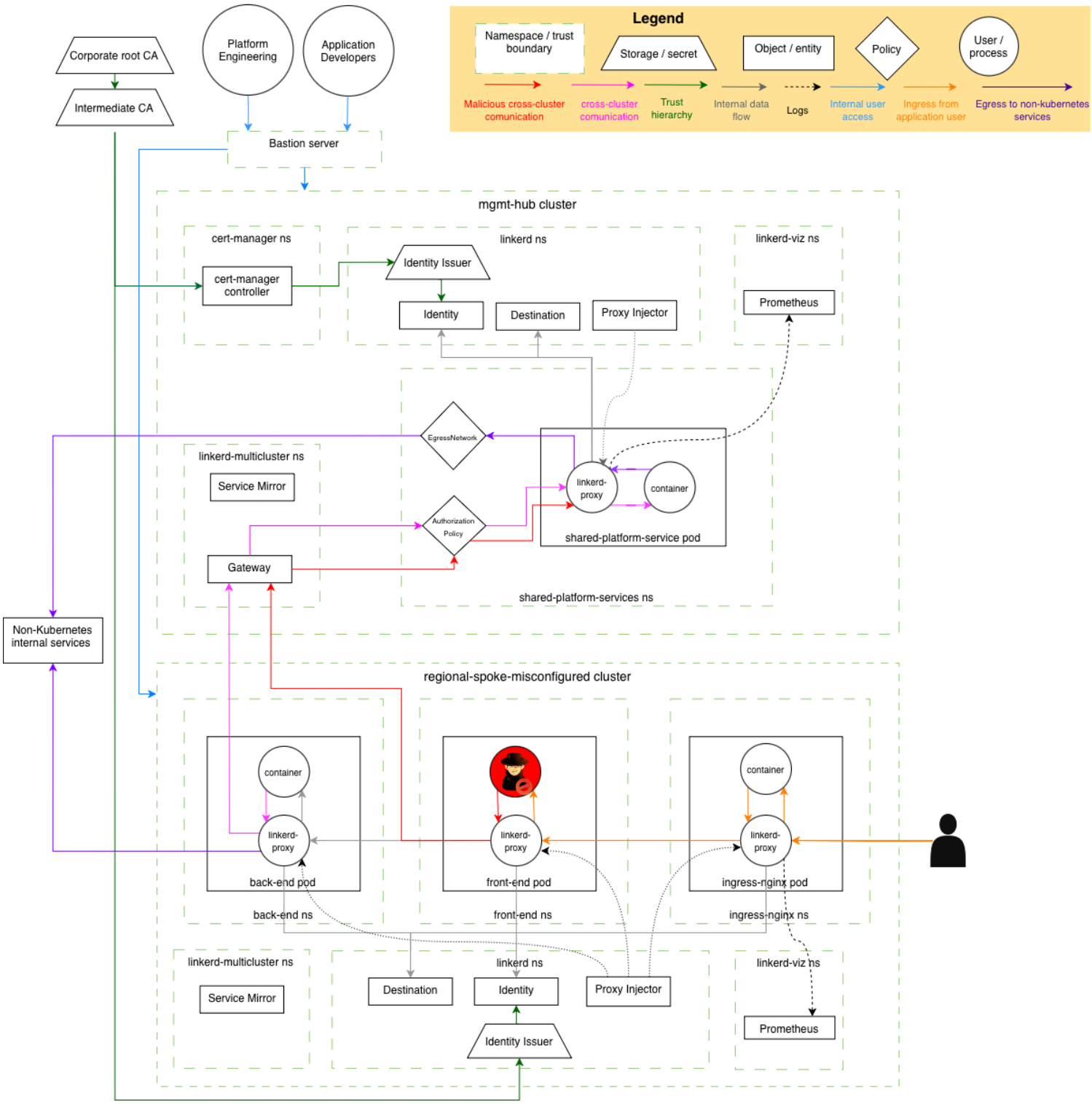
Linkerd provides granular defense-in-depth via `Server` and `AuthorizationPolicy` resources. Unlike L3/L4 `NetworkPolicies` (IP/Port), Layer 7 Linkerd policies can enforce:

- **Identity-Based access:** Can be used to restrict access to specific Kubernetes service accounts or workload identities associated with a given namespace
- **Method-Based access:** Can restrict the specific methods permitted in requests to a given service

However, in architectures utilising Linkerd's hierarchical multi-cluster communications mode, destination workloads see the identity of the destination cluster's gateway, impeding the granular enforcement of authorisation policies across cluster boundaries.

Data flow diagram

The Data Flow Diagram (DFD) below describes the flow of data between the various processes, entities and data stores in the system, as well as the trust boundaries between components.


Legend
Namespace / trust boundary
Storage / secret
Object / entity
Policy
User / process
Malicious cross-cluster comunication
cross-cluster comunication
Trust hierarchy
Internal data flow
Logs
Internal user access
Ingress from application user
Egress to non-kubernetes services
Corporate root CA
Intermediate CA
Platform Engineering
Application Developers
Bastion server
mgmt-hub cluster
cert-manager ns
cert-manager controller
linkerd ns
Identity Issuer
Identity
Destination
Proxy Injector
linkerd-viz ns
Prometheus
EgressNetwork
linkerd-multicluster ns
Service Mirror
Authorization Policy
Gateway
linkerd-proxy
container
shared-platform-service pod
shared-platform-services ns
Non-Kubernetes internal services
regional-spoke-misconfigured cluster
back-end pod
front-end pod
ingress-nginx pod
back-end ns
front-end ns
ingress-nginx ns
linkerd-multicluster ns
Service Mirror
linkerd ns
Destination
Identity
Proxy Injector
Identity Issuer
linkerd-viz ns
Prometheus

Flat network multi-cluster architecture overview

This section describes a multi-cluster Linkerd service mesh deployment implemented using a flat topology and federated services. This infrastructure is intended to be representative of a geographically distributed, regulated financial services organisation, in which:

- A global platform engineering function maintains control over multiple management clusters for resiliency. Federated services are used to load balance traffic from the regional spokes across the management clusters
- Application development teams are provided with regional clusters, which operate semi-autonomously, hosting jurisdiction-specific workloads while utilising shared services from the management clusters

This multi-cluster model enables segregation of duties and jurisdictional isolation, consistent with regulatory expectations for defense-in-depth and blast-radius containment.

Clusters and trust boundaries

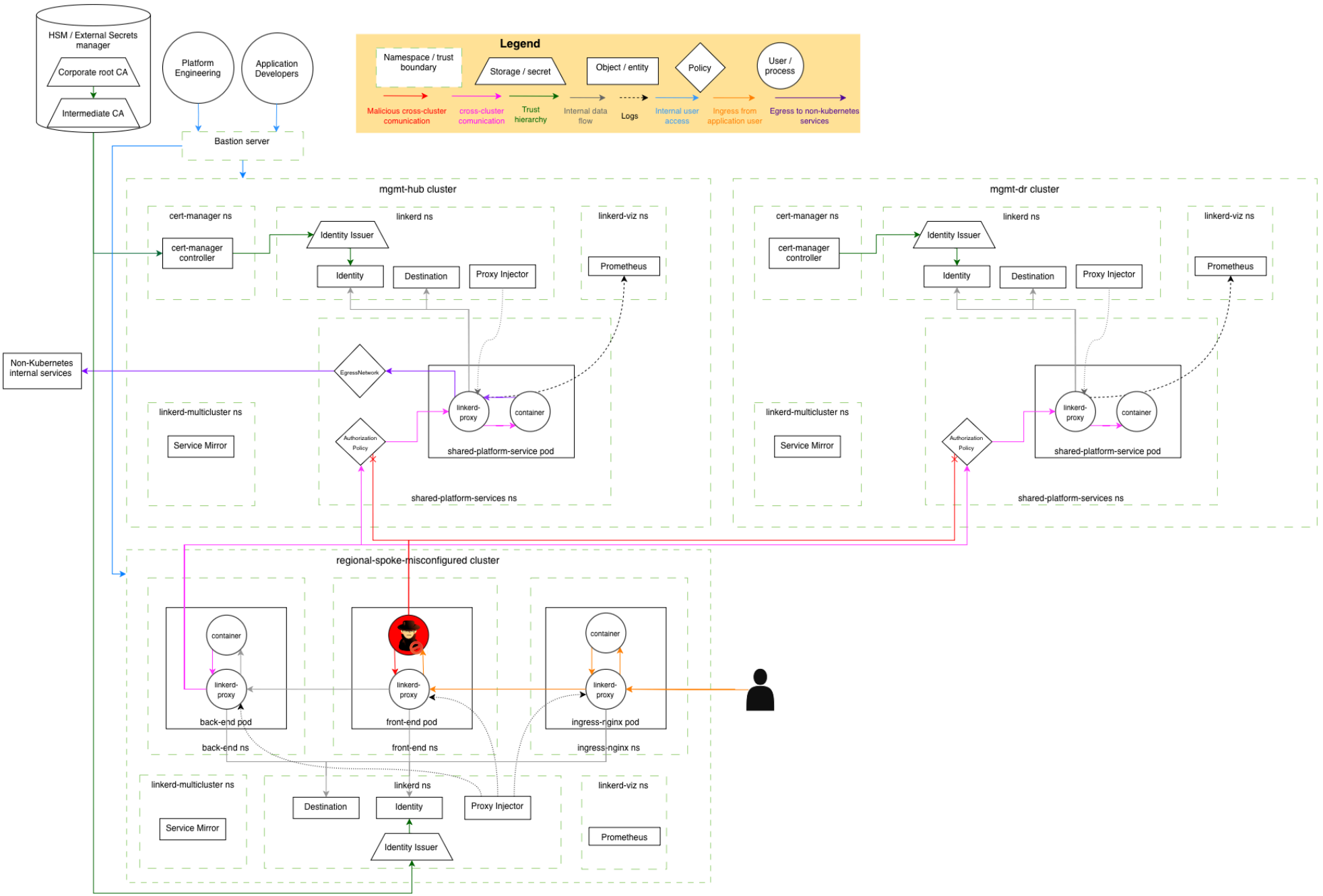
The architecture consists of the following three Kubernetes clusters:

- `mgmt-prod`: Represents a highly restricted administrative security zone. This cluster provides centrally governed platform services and data consumed by regional clusters. It is accessed exclusively by platform SRE teams
- `mgmt-dr`: An identical replica of the mgmt-prod cluster, primarily used for disaster recovery and providing additional capacity to federated services when required
- `regional-spoke-misconfigured`: Represents an organisation's deployments within a specific region. This cluster operates an independent Kubernetes control plane, hosts application workloads from various isolated tenants and is accessed by both internal application developers and external application users. It consumes shared services and data from the `mgmt-hub` cluster and uses an intentionally flawed Linkerd configuration to demonstrate the impact of an insecure service mesh

Aside from the Linkerd-specific issues affecting the `regional-spoke-misconfigured` cluster, the remainder of the entire multi-cluster infrastructure is considered to be hardened in alignment with security baselines, such as NIST 800-53 rev 5.

Data flow diagram

The Data Flow Diagram (DFD) below describes the flow of data between the various processes, entities and data stores in the system, as well as the trust boundaries between different users, clusters and network interfaces.


HSM / External Secrets manager
Corporate root CA
Intermediate CA
Platform Engineering
Application Developers
Bastion server
Legend
Namespace / trust boundary
Storage / secret
Object / entity
Policy
User / process
Malicious cross-cluster comunication
cross-cluster comunication
Trust hierarchy
Internal data flow
Logs
Internal user access
Ingress from application user
Egress to non-kubernetes services
mgmt-hub cluster
cert-manager ns
cert-manager controller
linkerd ns
Identity Issuer
Identity
Destination
Proxy Injector
linkerd-viz ns
Prometheus
Non-Kubernetes internal services
EgressNetwork
linkerd-multicluster ns
Service Mirror
Authorization Policy
linkerd-proxy
container
shared-platform-service pod
shared-platform-services ns
mgmt-dr cluster
regional-spoke-misconfigured cluster
back-end pod
front-end pod
ingress-nginx pod
back-end ns
front-end ns
ingress-nginx ns

Data

This section outlines the storage and management of sensitive data related to Linkerd across all three in-scope architectures. It covers critical assets such as trust anchor and identity issuer private keys, Linkerd policy Custom Resources (CRs), and multicluster link credentials. The section also introduces a data dictionary, which categorises data according to its confidentiality, integrity, and availability, using a priority legend for quick reference.

Data dictionary

Priority legend

Red	High
Amber	Medium
Green	Low

Data Name / Type	Notes	Confidentiality	Integrity	Availability
Trust Anchor Private Key	The trust anchor CA key used to sign the Linkerd identity issuer certificate. Often kept offline or in a secure vault	High	High	High
Identity Issuer Private Key	The private key used by the Linkerd Identity service to issue short-lived proxy certificates	High	High	High
Proxy Ephemeral Private Keys	The short-lived private key used by the linker sidecar to secure its connections	High	High	High
Service Account Tokens	Kubernetes `ServiceAccount` tokens used by the Linkerd proxy against the Identity service to request certificates	High	High	High
Application Traffic	The actual payload data (HTTP, gRPC, TCP) flowing between meshed services	High	High	High

Multicluster Link Credentials	Secrets containing `ServiceAccount` tokens used by the Service Mirror controller to read from target clusters	High	High	Medium
Core Mesh Config	Global and namespace-level configurations (e.g., `default-inbound-policy`) that dictate the baseline behaviour of the mesh	Medium	High	High
Trust Anchor Certificate	The public root certificate used to verify the Identity Issuer. Distributed to the cluster to establish the root of trust	Low	High	High
Identity Issuer Certificate	The public certificate of the issuer (signed by the Trust Anchor) used by proxies to validate their peers' certificates	Low	High	High
Proxy Ephemeral Certificates	The short-lived public certificate (typically valid for 24 hours) presented to peers during the mTLS handshake	Low	High	High
Proxy Injector Mutating Webhook Configuration	The Kubernetes resource that instructs the API server to send `Pod` creation requests to the Linkerd `proxy-injector` service	Low	High	High
Webhook Certificates	TLS certificates for Linkerd Mutating and Validating Admission Webhooks (`proxy-injector`, `sp-validator`, `policy-validator`)	Low	High	High
Server and Route CRs	`Server`, `HTTPRoute`, and `GRPCRoute` CRs that define which ports receive traffic and how L7 traffic is classified	Medium	High	Medium

Authorisation Policy CRs	Policy resources (`AuthorizationPolicy`, `MeshTLSAuthentication`, `NetworkAuthentication`) that enforce zero-trust rules, determining exactly which authenticated identities or networks can access a `Server` or route	Medium	High	Medium
Egress Policy CRs	`EgressNetwork` and `ExternalWorkload` CRs managing non-Kubernetes mesh expansion and tightly controlling outbound cluster traffic	Medium	High	Medium
`Link` CRs	The `Link` CRD configures how the clusters connect, specifying the gateway IP and the remote gateway's expected identity	Low	High	Medium
Linkerd Data Plane Annotations	`Pod` or `Namespace` annotations that override default sidecar behaviour (e.g., `skip-inbound-ports`, `proxy-cpu-limit`) or dictates whether the proxy should be injected (e.g., `linkerd.io/inject: enabled`)	Low	High	Medium
Telemetry & Logs	Prometheus metrics, access logs, and diagnostic `tap` captures containing request metadata and performance data	Medium	Medium	Low

Key Threats and Recommendations

This section details the specific findings and relevant recommendations from the threat modeling process.

Severity

The level of risk associated with each threat has been labelled using the following scoring system:

Red	High
Amber	Medium
Green	Low

However, the severity and recommended remediation for specific threats may be adjusted based on the business context, regulatory requirements, and the classification of data in a given deployment.

Categories

Threats are organized into the following primary categories:

- **Secrets and Key Management:** Threats related to the confidentiality, integrity, or availability of sensitive data used by Linkerd, including trust anchor and identity issuer private keys, multicluster link credentials, and webhook TLS certificates
- **Linkerd Control Plane:** Threats arising from insecure or suboptimal configuration of Linkerd control plane core components such as `linkerd-identity`, `linkerd-destination`, and `proxy-injector`
- **Linkerd Data Plane:** Threats arising from insecure or suboptimal configuration of Linkerd proxy sidecar
- **Linkerd Multicluster:** Threats related to Linkerd multicluster communications
    a. **Hierarchical Mode:** Threats specific to Linkerd's hierarchical (gateway) multicluster mode
    b. **Flat Network Mode:** Threats specific to Linkerd's flat network (peer-to-peer) multicluster mode
- **Linkerd Policy Configuration:** Threats arising from misconfigured or overly permissive Linkerd authorisation or egress policy resources such as `AuthorizationPolicy`, `MeshTLSAuthentication`, `NetworkAuthentication`, `Server`, `HTTPRoute`, `GRPCRoute`, or `EgressNetwork`
- **Linkerd Policy Integrity:** Threats arising from the unauthorised modification or deletion of Linkerd authorisation and egress policy resources

- **RBAC:** Threats stemming from overly permissive Role-Based Access Control assignments
- **Network:** Threats related to insufficient network isolation, uncontrolled ingress/egress, or exposure of Linkerd components to other workloads
- **Admission Control:** Threats arising from insecure or suboptimal configuration of Linkerd's mutating and validating admission webhook configurations, or `failurePolicy` settings
- **Kubernetes:** Threats associated with underlying Kubernetes platform or API misconfigurations that affect Linkerd security
- **Default Configuration:** Threats arising from Linkerd default behavior as configured in the default Helm chart values
- **Ingress:** Threats associated with external traffic entering the mesh, including misconfigured ingress controllers and the failure to sanitise Linkerd-specific routing headers
- **Linkerd Viz:** Threats arising from insecure or suboptimal configuration of the `linkerd-viz` extension

Threats

This section presents the security threats that were identified during the threat modeling process. Each threat is assigned a unique ID for reference and includes a description of the threat scenario and potential misconfigurations causing it, along with potential mitigation strategies and recommendations.

T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Low	Secrets and Key Management RBAC Default Configuration

Impact: An attacker can forge issuer or workload identity certificates, allowing them to impersonate any service within the mesh.

Description: Malicious or compromised actors with sufficient Kubernetes Role-Based Access Control (RBAC) privileges can extract the private keys for either the trust anchor or the identity issuer stored as Kubernetes secrets in the cluster. Extraction of these keys undermines the cryptographic root of trust for the service mesh, allowing the attacker to mint arbitrary identity certificates for any workload.

While the extraction of either key undermines the cryptographic root of trust for the service mesh, the operational impact and remediation effort differ:

- **Identity Issuer Key:** Compromising this intermediate certificate allows an attacker to mint valid workload certificates to impersonate any meshed service and bypass mTLS authentication boundaries
- **Trust Anchor Key:** Compromising the trust anchor breaks the foundational security model of the mesh. An attacker can mint new issuer certificates and workload identity leaf certificates for the remaining lifespan of the trust anchor certificate (usually long-lived). Remediating a compromised trust anchor requires a complete rollover of all cryptographic material and proxy sidecars

In multi-cluster environments, the compromise of either key also has cross-cluster consequences because linked clusters share the same root trust anchor.

Recommendations:

- Store the trust anchor private key externally in a Hardware Security Module (HSM) or a dedicated secrets management system (e.g., cloud Key Management Service (KMS), OpenBao, or HashiCorp Vault) to prevent in-cluster extraction
- Use short-lived identity issuer certificates and implement automated rotation, for example with `cert-manager`. This minimises the window of exposure if an issuer key is compromised, though it provides no mitigation if the root trust anchor itself is leaked

- Implement certificate transparency logging to detect the unauthorised issuance of certificates
- Enable `etcd` encryption at rest to protect `Secret` resources containing private keys
- Enforce the principle of least privilege for RBAC permissions concerning the `linkerd` namespace, limiting access to `Secret` resources
- Enable Kubernetes audit logging and configure alerting for anomalous read operations on `Secrets` within the `linkerd` namespace
- Document and regularly test incident response playbooks for emergency certificate rotation procedures

References:

- https://linkerd.io/2.20/features/automatic-mtls/
- https://linkerd.io/2.20/tasks/automatically-rotating-control-plane-tls-credentials/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1552.007: Unsecured Credentials: Container API T1552.004: Unsecured Credentials: Private Keys T1649: Steal or Forge Authentication Certificates	AC-3: Access Enforcement AC-6: Least Privilege SC-12: Cryptographic Key Establishment and Management SC-13: Cryptographic Protection SC-17: Public Key Infrastructure Certificates SC-28: Protection of Information at Rest IR-8: Incident Response Plan

T-02 - Unauthorised access to meshed services due to permissive default inbound policy

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Low	Default Configuration Linkerd Policy Configuration

Impact: Meshed services without an explicitly defined `Server` CRs accept plaintext traffic from unmeshed clients, bypassing the mTLS trust model.

Description: During a Linkerd install, the `proxy.defaultInboundPolicy` field is used to specify the cluster-wide default policy which controls what traffic is allowed to meshed pods. This cluster-wide default can be overridden for specific resources by setting the annotation `config.linkerd.io/default-inbound-policy` on either a pod spec or its namespace.

If the default inbound policy is set to `all-unauthenticated` , which is the default value in the Linkerd Helm Chart, or `audit` (same as `all-unauthenticated` but requests get flagged in logs and metrics), the proxy will permit all inbound traffic on application ports that lack explicitly defined `Server` CR, allowing plaintext connections from clients without a valid mTLS identity.

Recommendations:

- Deploy Linkerd with `proxy.defaultInboundPolicy` set to a restrictive value
    - Ideally, set the default policy to `deny`. This enforces a strict model where all incoming traffic is dropped unless an explicit `Server` resource is defined and a corresponding `AuthorizationPolicy` or the `Server`'s `accessPolicy` (applied to traffic not matching any associated authorisation policies, set to `deny` by default) allow it
    - Alternatively, use `cluster-authenticated` (allows traffic only from meshed clients possessing a valid identity within the same cluster) or `all-authenticated` (allows any meshed client with a valid identity), e.g. as a temporary measure until granular authorisation policies are universally rolled out
- Define explicit `Server` resources for all exposed application ports and map them to strict `AuthorizationPolicy` definitions, ensuring least-privilege access regardless of the global default
- Utilise `MeshTLSAuthentication` resources within the authorisation policies to restrict access further, explicitly allowing connections only from the specific client identities required by the application

- Use an admission controller (e.g. Kyverno or OPA Gatekeeper) to ensure that the cluster-wide default policy isn't overridden in pod or namespace specs via the `config.linkerd.io/default-inbound-policy` annotation and set to a more permissive value
- Enforce least privilege RBAC for resources in the `linkerd` namespace to reduce the risk of tampering with the cluster-wide default inbound policy configuration

References:

- https://linkerd.io/2.20/features/automatic-mtls/#caveats
- https://linkerd.io/2.20/reference/authorization-policy/#default-policies
- https://linkerd.io/2.20/tasks/restricting-access/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1210: Exploitation of Remote Services	CM-2: Baseline Configuration CM-6: Configuration Settings AC-4: Information Flow Enforcement SC-8: Transmission Confidentiality and Integrity SC-23: Session Authenticity

T-03 - Unmeshed workload admission due to permissive proxy injector webhook failure policy

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Medium	Default Configuration Admission Control

Impact: Pods can be deployed in the cluster without the Linkerd proxy sidecar, allowing them to bypass mTLS authentication, authorisation policies, and all mesh-level security controls. Additionally, requests from unmeshed pods will be rejected by services with strict inbound policies, causing potential outages.

Description: The `failurePolicy` field on `MutatingWebhookConfiguration` resources controls Kubernetes API server behaviour when a webhook call fails to complete. This could occur when the webhook endpoint is unreachable, returns an error, or exceeds the configured timeout. When `failurePolicy` is set to `Ignore`, the API server admits the request as if the webhook were not configured. When set to `Fail`, the API server rejects the request.

If the failure policy of the Linkerd proxy injector webhook is set to `Ignore` (which is the default value for `webhookFailurePolicy` in the Linkerd Helm Chart), unavailability of the proxy injector would cause the Kubernetes API server to bypass proxy injection and schedule pods without the Linkerd sidecar, allowing these pods to bypass mTLS authentication, authorisation policies, and all mesh-level security controls. Furthermore, this introduces an availability risk: if these unmeshed pods attempt to communicate with strictly configured meshed services, their traffic will be dropped for lacking a valid mTLS identity.

Recommendations:

- Ensure that the proxy injector's `MutatingWebhookConfiguration` has `failurePolicy` set to `Fail` (configured via `webhookFailurePolicy: Fail` in the Linkerd Helm Chart)
- Apply `NetworkPolicy` resources to only allow ingress traffic to the Linkerd proxy injector webhook from the Kubernetes API server, preventing compromised internal cluster workloads from executing a DoS attack against the webhook endpoint
- Deploy multiple replicas of the Linkerd proxy injector and configure pod anti-affinity rules and a Pod Disruption Budget (PDB) to ensure high availability and minimise the risk of webhook downtime. This can be done by deploying the Linkerd control plane using the High Availability (HA) Helm Chart or through manual configuration

- Implement admission controller policies (e.g. using Kyverno or OPA Gatekeeper) to reject pod creation requests in designated meshed namespaces if they lack the required Linkerd proxy containers
- Enforce least privilege RBAC for `MutatingWebhookConfiguration` resources to minimise the risk of tampering with the webhook's configuration

References:

- https://linkerd.io/2.20/features/ha/#proxy-injector-failure-policy
- https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#failure-policy

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1562.001: Impair Defenses: Disable or Modify Tools T1610: Deploy Container	CM-2: Baseline Configuration CM-6: Configuration Settings AC-4: Information Flow Enforcement SC-7: Boundary Protection SC-8: Transmission Confidentiality and Integrity SC-23: Session Authenticity

T-04 - Overly permissive or absent authorisation policies

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Medium	Linkerd Policy Configuration

Impact: Overly permissive or absent authorisation policies may allow an attacker to leverage the valid identity of a compromised low-trust workload to communicate with other internal services, expanding the blast radius of the attack and potentially enabling lateral movement.

Description: While mTLS ensures that traffic is encrypted and identities are cryptographically verified, it does not enforce access control on its own. If authorisation policies are not created for meshed workloads, or if existing policies are overly broad, the mesh fails to enforce a zero-trust model.

For example, a `MeshTLSAuthentication` CR configured to allow any authenticated mesh identity (`*`), or a `NetworkAuthentication` CR permitting large CIDR blocks, effectively grants cluster-wide access. In these conditions, if a low-trust workload is compromised, the attacker can leverage its valid identity to communicate with internal services that lack explicitly defined, least-privilege boundaries.

Recommendations:

- Set a restrictive `proxy.defaultInboundPolicy` at the cluster level (ideally `deny`, or alternatively `cluster-authenticated` or `all-authenticated` as a temporary measure until explicit authorisation policies are universally rolled out)
- Define explicit `Server` resources for all exposed application ports
- Enforce route-level authorisation by integrating `HTTPRoute` and `GRPCRoute` CRs
- Implement fine-grained `AuthorizationPolicy` CRs that map to those `Server` or route resources
- Pair policies with specific authentication targets: use `MeshTLSAuthentication` to restrict access to approved identities, and `NetworkAuthentication` to restrict access to exact IP addresses or narrow subnets

References:

- https://linkerd.io/2.20/reference/authorization-policy/
- https://linkerd.io/2.20/tasks/restricting-access/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1562.001: Impair Defenses: Disable or Modify Tools T1556.009: Modify Authentication Process: Conditional Access Policies	AC-4: Information Flow Enforcement CM-6: Configuration Settings SC-23: Session Authenticity

T-05 - Unauthorised modification of Linkerd policy resources

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	High	RBAC Linkerd Policy Integrity

Impact: Principals with RBAC permissions to modify Linkerd policy resources can dismantle the mesh's zero-trust boundaries, allowing unauthorised access to services or enabling data exfiltration.

Description: Linkerd relies on Custom Resources (CRs) such as `AuthorizationPolicy`, `MeshTLSAuthentication`, `NetworkAuthentication`, `Server`, `HTTPRoute`, and `GRPCRoute`, to control which types of traffic are allowed to a meshed pod, and `ExternalNetwork` to control traffic that leaves the cluster. Principals with Kubernetes RBAC privileges allowing the creation, modification, or deletion of these resources can subvert the mesh's security posture. This could involve configuring overly permissive policies that expose internal services to unauthenticated clients or unintended identities, authorising egress traffic to malicious external infrastructure, or deleting legitimate policies to break application connectivity.

The impact of such tampering is amplified in hierarchical multi-cluster architectures if policy resources related to the Linkerd gateway are affected. Because the Linkerd gateway acts as the central ingress point for inbound traffic from remote clusters and may be internet-facing, altering its associated `Server` and authorisation policies could expose internal services to external attackers or unintended remote cluster identities, or cause cross-cluster routing outages.

Recommendations:

- Enforce least privilege RBAC for Linkerd policy CRs. Implement separation of duties between platform administrators (who define security baselines) and application teams (who define service-specific access)
- Implement GitOps-based configuration management (e.g., using FluxCD or ArgoCD) to deploy policy resources as declarative zero-trust baselines. Enable automated drift detection to identify and revert any unauthorised degradation of these policies
- Use an admission controller (e.g. Kyverno or OPA Gatekeeper) to prevent the application of policies that violate organisational zero-trust standards (e.g., rejecting policies that use wildcards or allow unauthenticated access)
- Enable Kubernetes API audit logging and configure SIEM alerting for anomalous modifications of policy resources that indicate an attempt to weaken the mesh's security posture

References:

- https://linkerd.io/2.20/reference/authorization-policy/
- https://linkerd.io/2.20/reference/egress-network/
- https://kubernetes.io/docs/reference/access-authn-authz/rbac/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1552.007: Unsecured Credentials: Container API T1562.001: Impair Defenses: Disable or Modify Tools T1556.009: Modify Authentication Process: Conditional Access Policies	AC-3: Access Enforcement AC-6: Least Privilege CM-2: Baseline Configuration CM-5: Access Restrictions for Change

T-06 - Identity spoofing via Kubernetes Service Account token theft or RBAC abuse

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Low	Kubernetes RBAC

Impact: By exchanging a stolen or fraudulently obtained Kubernetes Service Account token for a valid Linkerd identity certificate, an attacker can impersonate a legitimate mesh workload and use its identity to access restricted workloads.

Description: Linkerd's control plane issues mTLS identity certificates by validating Kubernetes Service Account (SA) tokens presented by proxy sidecars. If an attacker compromises a pod and extracts its SA token, they can present this token to the Linkerd `identity` service and exchange it for a valid certificate tied to that workload's identity and impersonate it.

Furthermore, an attacker possessing certain Kubernetes RBAC permissions within a namespace can impersonate any service account within that boundary, for example by creating new workloads with arbitrary service accounts assigned, creating Kubernetes `Secrets` annotated with a `kubernetes.io/service-account.name` annotation to extract long-lived tokens, or directly interacting with the `TokenRequest` API (`create serviceaccounts/token`).

Compounding this risk, whilst Linkerd uses bounded service account tokens with a specific audience (`identity.l5d.io`), at the time of writing it falls back to validating legacy, unbounded default service account tokens upon encountering audience validation errors, increasing the viability of using generic stolen tokens for identity forgery.

Recommendations:

- Utilise Kubernetes projected service account volume mounts (Service Account Token Volume Projection) to enforce bound tokens with specific audiences and short Time-To-Live (TTL) values, minimising the exposure window of a stolen token
- Disable the automatic mounting of default service account tokens (`automountServiceAccountToken: false`) at the pod or service account specification level for all workloads, unless the application requires interaction with the Kubernetes API

- Enforce Kubernetes RBAC least-privilege principles. Restrict permissions to create workloads (`Pod`, `Deployment`, and others), manage `Secrets`, or access the `serviceaccounts/token` subresource within application namespaces
- Implement robust namespace segregation, deploying workloads with differing trust levels and risk profiles into separate Kubernetes namespaces to contain the blast radius of a compromised namespace-scoped RBAC role

References:

- https://linkerd.io/2.20/features/automatic-mtls/#how-does-linkerds-mtls-implementation-work
- https://linkerd.io/2021/12/28/using-kubernetess-new-bound-service-account-tokens-for-secure-workload-identity/
- https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1552: Unsecured Credentials T1552.007: Unsecured Credentials: Container API T1528: Steal Application Access Token T1649: Steal or Forge Authentication Certificates	CM-2: Baseline Configuration CM-6: Configuration Settings AC-6: Least Privilege CM-7: Least Functionality AC-3: Access Enforcement

T-07 - Ingress mode proxy open relay via spoofed `l5d-dst-override` header

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Low	Ingress Linkerd Data Plane

Impact: External attackers can bypass ingress routing rules, turning the ingress-mode Linkerd proxy into an open relay. This allows them to execute Server-Side Request Forgery (SSRF) style attacks to arbitrary in-cluster or external endpoints.

Description: When integrating external ingress controllers, the Linkerd proxy is often deployed in ingress mode using the `linkerd.io/inject: ingress` annotation. In this mode, the proxy relies on the `l5d-dst-override`, `Host` or `:authority` HTTP headers to determine the target routing destination. If the external ingress controller is misconfigured and fails to strip or sanitise the `l5d-dst-override` header from incoming requests, an attacker can supply a custom destination within the header and coerce the Linkerd sidecar into acting as an open relay, circumventing the intended ingress routing and routing requests to arbitrary in-cluster or external endpoints.

Recommendations:

- Only set the `linkerd.io/inject: ingress` annotation if this is required by the ingress controller's routing behaviour. If the ingress controller routes traffic directly to the IP and port of the `Service`, use standard proxy injection to eliminate the proxy's reliance on the `l5d-dst-override` header
- When ingress mode is required for the proxy, configure the ingress controller to strip or overwrite the `l5d-dst-override` header from all inbound client requests
- Ensure the ingress controller only populates the `l5d-dst-override` header using trusted, internally derived routing values
- Implement Linkerd `AuthorizationPolicy` resources to restrict the internal destinations that the ingress controller's identity is permitted to access, thereby limiting the blast radius if header spoofing occurs

References:

- https://linkerd.io/2.20/tasks/using-ingress/
- https://linkerd.io/2.20/reference/proxy-configuration/#ingress-mode
- https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1190: Exploit Public-Facing Application	AC-4: Information Flow Enforcement SC-7: Boundary Protection SI-10: Information Input Validation

T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Medium	Linkerd Data Plane Kubernetes

Impact: Bypassing the Linkerd proxy sidecar or preventing its injection removes the workload from the mesh's security boundaries. This allows network traffic to flow in plaintext, circumventing mTLS encryption, evading L7 authorisation policies, and remaining invisible to telemetry. Furthermore, destinations enforcing mTLS will drop this unauthenticated traffic, causing availability disruptions.

Description: The Linkerd data plane relies on `iptables` rules within the pod's network namespace to intercept and route traffic through the proxy sidecar. The mesh's security controls can be subverted through several pod-level misconfigurations, privilege escalations, or annotation abuse:

- **Host Networking:** Workloads deployed with `hostNetwork: true` share the host node's network namespace. Because intercepting host-level traffic is unsafe, Linkerd does not support injection for these workloads, meaning the proxy sidecar is never injected at all
- **Excessive Linux Capabilities:** If an application container is granted excessive capabilities, specifically `NET_ADMIN` and `NET_RAW`, an attacker with a foothold on the pod can alter the `iptables` rules within the pod's network namespace to disable or circumvent the proxy redirection
- **Proxy UID Collision:** To prevent infinite routing loops, Linkerd's `iptables` configuration includes an `ignore-proxy-uid` rule ensuring that outbound packets generated by the proxy's Linux User ID (defaulting to 2102) skip redirection. If an application container within the same pod is configured to execute under the same UID, its traffic bypasses the sidecar. Furthermore, if a compromised application container runs as `root` or possesses the `CAP_SETUID` capability, an attacker can intentionally switch to the proxy's UID to evade outbound proxy redirection
- **Annotation Tampering:** If an attacker has RBAC permissions to modify `Pod` or `Namespace` resources, they can manipulate Linkerd-specific annotations. This allows them to disable sidecar injection (`linkerd.io/inject: disabled`), configure the proxy to ignore specific traffic via skipped ports or CIDR ranges, or replace the legitimate proxy with a compromised, attacker-controlled container image

Recommendations:

- Enable the Linkerd Container Network Interface (CNI) plugin (`cniEnabled: true`). This delegates `iptables` configuration to a `DaemonSet`, eliminating the need for the `linkerd-init` container to require the `NET_ADMIN` capability
- Enforce Kubernetes Pod Security Standards (PSS) at the namespace level or implement equivalent admission controller policies. Explicitly drop all unnecessary Linux capabilities (`securityContext.capabilities.drop: ["ALL"]`) for application containers to eliminate the risk of `NET_ADMIN` and `NET_RAW` abuse. Disallow workloads from requesting host networking (`hostNetwork: true`)
- Enforce strict `securityContext` configurations on all application containers, specifically setting `runAsNonRoot: true`, and ensuring `runAsUser` cannot be set to the reserved proxy UID (2102, or the custom UID specified via the `config.linkerd.io/proxy-uid` annotation)
- Enforce the principle of least privilege for Kubernetes RBAC, limiting permissions to create or modify workloads (`Pods`, `Deployments`, and others) and `Namespaces`. Manage workload templates via version-controlled GitOps pipelines with automated drift detection
- Use an admission controller (e.g., Kyverno or OPA Gatekeeper) to enforce constraints on Linkerd annotations. Enforce image allowlisting to reject unapproved container images for the proxy
- Implement runtime security monitoring (e.g., using Falco or Tetragon) to detect and alert upon unauthorised `iptables` rule modifications, anomalous network syscalls (such as raw socket creation), or unexpected UID transitions within application containers

References:

- https://linkerd.io/2.20/reference/iptables/
- https://linkerd.io/2.20/features/cni/
- https://github.com/linkerd/linkerd2/discussions/11040#discussioncomment-6256564
- https://linkerd.io/2.20/reference/proxy-configuration/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1610: Deploy Container T1562.001: Impair Defenses: Disable or Modify Tools T1068: Exploitation for Privilege Escalation	CM-2: Baseline Configuration CM-6: Configuration Settings CM-7: Least Functionality SC-3: Security Function Isolation

	SC-39: Process Isolation SI-3: Malicious Code Protection SI-4: System Monitoring

T-09 - Linkerd control plane compromise

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	High	Linkerd Control Plane RBAC

Impact: An attacker who compromises the Linkerd control plane can dismantle the mesh's security model. Depending on the compromised control plane component, they may issue fraudulent workload identities, maliciously reroute internal traffic, inject compromised proxy sidecars, or disable security policies.

Description: The Linkerd control plane operates as the central authority of the service mesh, comprising several components: the `identity` service, `destination` service, and the `proxy-injector`. If an attacker gains unauthorised access to the pods, service accounts, or underlying nodes hosting these components, whether through the exploitation of a runtime vulnerability, lateral movement within the cluster, or a software supply chain attack (e.g., pulling a tampered container image from a compromised registry), they achieve absolute control over the mesh's data plane operations. Compromising the `identity` service allows the issuance of cryptographically valid mTLS certificates for any arbitrary identity. Exploiting the `destination` service enables the attacker to manipulate service discovery and policy distribution. Furthermore, compromising the `proxy-injector` allows the attacker to inject malicious sidecar containers into newly scheduled application pods, or selectively bypass injection entirely to force plaintext communication.

Recommendations:

- Verify the cryptographic signatures of Linkerd control plane images (using tools like Cosign/Sigstore) prior to deployment to ensure authenticity and integrity. Deploy validating admission controllers (e.g., Kyverno or OPA Gatekeeper) to automatically reject any unsigned or modified control plane images
- Pull Linkerd images from a hardened, internal container registry that performs continuous vulnerability scanning, rather than relying on public internet registries
- Apply the principle of least privilege to Kubernetes RBAC roles governing access to the control plane namespaces. Limit which users, groups, or CI/CD pipelines hold permissions to read, create, modify, delete, or execute commands (`pods/exec` subresource) within the `linkerd` namespace to prevent unauthorised tampering with control plane pods, secrets, or configurations
- Enforce the principle of least privilege for the Kubernetes RBAC permissions assigned to the control plane's own Service Accounts, ensuring they cannot be abused to modify cluster-wide resources beyond their required operational scope

- Apply Kubernetes node taints, tolerations, and node affinity rules to schedule Linkerd control plane components on dedicated, highly restricted nodes. This physical or logical separation prevents lateral movement from compromised, lower-trust application workloads running on shared worker nodes
- Implement monitoring, alerting, and anomaly detection for the control plane components. Continuously analyse audit logs and Linkerd telemetry for anomalous certificate issuance requests, unexpected internal routing updates, or unauthorised modifications to the mutating webhook configuration

References:

- https://linkerd.io/2.20/reference/architecture/#control-plane
- https://kubernetes.io/docs/reference/access-authn-authz/rbac/
- https://github.com/sigstore/cosign
- https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1552.007: Unsecured Credentials: Container API T1195.002: Supply Chain Compromise: Compromise Software Supply Chain	AC-3: Access Enforcement AC-6: Least Privilege CM-5: Access Restrictions for Change SI-4: System Monitoring SI-7: Software, Firmware, and Information Integrity SR-4: Provenance

T-10 - Identity spoofing or denial of service via trust bundle tampering

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	High	Secrets and Key Management RBAC

Impact: Unauthorised modification of the mesh's shared trust bundle allows an attacker to inject a rogue Certificate Authority (CA) or delete the legitimate trust anchor. This enables mesh-wide identity spoofing and traffic interception, or mesh-wide DoS for all strict-mTLS inter-service communication.

Description: The Linkerd control plane distributes the mesh's shared mTLS trust anchor (`ca-bundle.crt`) to all sidecar proxies using the `linkerd-identity-trust-roots ConfigMap` located in the `linkerd` namespace. A principal with RBAC permissions to modify `ConfigMaps` in this namespace can tamper with the cryptographic root of trust. By injecting a malicious root CA into the bundle, the attacker can mint fake workload certificates that the proxy sidecars of newly scheduled pods will blindly trust. This bypasses identity-based authorisation policies and allows unauthorised access to sensitive pods. Alternatively, intentionally corrupting or deleting the legitimate trust bundle will cause newly scheduled sidecars to fail peer validation, resulting in inter-service traffic being mutually rejected and causing mesh-wide outages.

Recommendations:

- Limit Kubernetes RBAC permissions to ensure only authorised administrators, GitOps controllers, and automated PKI operators (like `cert-manager`) have `update`, `patch`, or `delete` privileges for `ConfigMaps` within the `linkerd` namespace
- Implement admission controller policies (e.g., using Kyverno or OPA Gatekeeper) to block manual modification of the `linkerd-identity-trust-roots ConfigMap`, allowing mutations exclusively from designated GitOps or PKI service accounts
- Enable and monitor Kubernetes API audit logs for unauthorised `PUT`, `PATCH`, or `DELETE` events targeting this specific `ConfigMap`, triggering alerts for suspicious activity
- Manage Linkerd configurations and trust anchors strictly through a GitOps workflow (e.g., Flux CD or Argo CD). Enable automated drift detection to identify and revert any unauthorised drift or tampering

References:

- https://linkerd.io/2.20/features/automatic-mtls/#how-does-linkerds-mtls-implementation-work

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1552.007: Unsecured Credentials: Container API T1553.004: Subvert Trust Controls: Install Root Certificate T1649: Steal or Forge Authentication Certificates T1499: Endpoint Denial of Service	AC-3: Access Enforcement AC-6: Least Privilege CM-5: Access Restrictions for Change

T-11 - Cross-cluster identity obfuscation via gateway confused deputy

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Low	Linkerd Multicluster - Hierarchical Mode Linkerd Policy Configuration

Impact: An attacker compromising a low-privilege workload in a source cluster can leverage the multicluster gateway to access sensitive, restricted services in a destination cluster due to the gateway obfuscating the request's original identity, breaking granular zero-trust enforcement and enabling lateral movement across cluster boundaries.

Description: In Linkerd's hierarchical multicluster deployment mode, inter-cluster traffic is routed through a dedicated `linkerd-gateway` component. When a workload in the destination cluster receives a cross-cluster request, it authenticates the mTLS connection against the local gateway's identity, losing the cryptographic context of the actual source pod in the remote cluster. Because the gateway acts as an intermediary for all incoming cross-cluster traffic, it effectively becomes a "confused deputy."

This identity obfuscation breaks the granular enforcement of Linkerd `AuthorizationPolicies` across cluster boundaries. A destination service cannot distinguish between a highly privileged administrative pod and a low-privilege frontend pod if both originate from the same remote cluster. Consequently, administrators may write overly permissive policies that allow the gateway identity broad access to internal services.

Furthermore, the default `linkerd-multicluster` Helm chart applies permissive policies to the gateway itself, allowing any remote identity to initiate connections if not explicitly hardened.

Recommendations:

- Replace the default permissive policies created by the `linkerd-multicluster` Helm chart for the `linkerd-gateway` component with strict `AuthorizationPolicy` resources. Restrict which remote cluster identities (via `MeshTLSAuthentication`) and source networks (via `NetworkAuthentication`) are permitted to communicate with the gateway
- Enforce authorisation policies on the destination cluster workloads to restrict access from the gateway identity. Combine this with specific `HTTPRoute` rules to expose only non-sensitive, necessary API paths to cross-cluster traffic
- Implement Linkerd egress policies and Kubernetes Network Policies in the source cluster, restricting which local workloads are permitted to route traffic to the remote destination cluster's gateway

- Enable access logging for the gateway proxy to log the original identity of cross-cluster requests
- Deploy multiple, dedicated gateways, logically segmented by trust level or application domain (e.g., routing sensitive traffic through a "high-trust-gateway" and general traffic through a "low-trust-gateway"). This isolates traffic flows, although it significantly increases manual configuration overhead and operational complexity
- Implement application-layer authorisation for sensitive services providing defence-in-depth when mesh-level identity is obfuscated by the gateway
- Consider migrating to Linkerd's flat network (peer-to-peer) multicluster deployment mode. In a flat network architecture, proxy sidecars establish direct mTLS connections across clusters, preserving the original cryptographic workload identity end-to-end and allowing for precise `AuthorizationPolicy` enforcement. However, this approach requires careful evaluation of its own security trade-offs, as it necessitates a fully routable network between cluster pod CIDRs, potentially increasing the blast radius of a network-level compromise. **Recommendation depends on the specific environment**

References:

- https://linkerd.io/2.20/features/multicluster/
- https://linkerd.io/2.20/reference/authorization-policy/
- https://en.wikipedia.org/wiki/Confused_deputy_problem
- https://github.com/linkerd/linkerd2/issues/7235#issuecomment-965829215
- https://linkerd.io/2.20/features/access-logging/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1133: External Remote Services T1021: Remote Services T1199: Trusted Relationship T1078: Valid Accounts	AC-3: Access Enforcement AC-4: Information Flow Enforcement SC-46: Cross Domain Policy Enforcement

T-12 - Cross-cluster lateral movement in flat network multicluster mode

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Medium	Linkerd Multicluster - Flat Network Mode Network

Impact: A localised compromise in a single cluster can cascade into a multi-cluster breach, expanding the blast radius of a breach.

Description: In Linkerd's flat network (peer-to-peer) multicluster deployment mode, proxy sidecars establish direct mTLS connections with their counterparts in remote clusters. This architecture mandates that pod IP addresses be directly routable across cluster boundaries, effectively merging the network planes of distinct clusters into a single flat network. Because the network-level isolation between clusters is dissolved, an attacker who compromises a pod in one cluster gains network reachability to the pod CIDR blocks of all connected clusters.

If multicluster-aware Kubernetes Network Policies or restrictive Linkerd authorisation policies and egress policies are not uniformly enforced, the compromised pod may be used as a staging ground to scan and laterally move into remote cluster environments. The inherent trust of the network layer combined with a lack of granular access controls increases the blast radius of a breach and enables cross-cluster exploitation.

Recommendations:

- Create default-deny Kubernetes `NetworkPolicies` in all namespaces across every connected cluster to ensure workloads can only communicate with allowlisted peers, whether local or remote
- Utilise advanced CNI plugins (such as Cilium with Cluster Mesh) capable of enforcing cross-cluster network policies that span the entire flat network topology, ensuring consistent enforcement regardless of the pod's cluster location
- Enforce strict Linkerd `AuthorizationPolicies` universally. Even if the underlying flat network allows the packets to route across clusters, the destination proxy sidecar must be configured to verify the remote source workload's mTLS identity

References:

- https://linkerd.io/2.20/features/multicluster/#flat-networks
- https://docs.cilium.io/en/stable/network/clustermesh/policy/
- https://linkerd.io/2.20/reference/authorization-policy/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1210: Exploitation of Remote Services	AC-4: Information Flow Enforcement SC-7: Boundary Protection SC-46: Cross Domain Policy Enforcement

T-13 - Remote cluster Kubernetes API access via link credentials theft

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	Low	Low	Linkerd Multicluster Secrets and Key Management RBAC

Impact: Compromising the multicluster link credentials grants an attacker access to a remote cluster's Kubernetes API. The risk scales with how broadly the credential's RBAC is scoped. With the default read-only cluster-wide permissions, an attacker can conduct extensive reconnaissance, mapping out the entire service and endpoint topology of the remote cluster.

Description: When establishing a multicluster link, the Service Mirror controller in the source cluster requires access to the remote (target) cluster's Kubernetes API to observe and mirror services. By default, the `linkerd-multicluster` Helm chart facilitates this by generating a long-lived Kubernetes Service Account token for the target cluster and storing it as a static `Secret` within the `linkerd` or `linkerd-multicluster` namespace of the source cluster.

If an attacker gains access to secrets in this local namespace, they can extract this long-lived credential. While the default RBAC permissions configured by the Helm chart restrict this token to read-only operations (`get`, `list`, and `watch` on `services`, `endpoints`, `endpointslices`, and `namespaces`), these permissions are bound via a `ClusterRole`. This means the token grants global visibility across the entire target cluster. An attacker can leverage this cluster-wide scope to map out sensitive backend services, even in namespaces that were never intended to be federated or exposed across the multicluster link.

Recommendations:

- If you only intend to mirror services from specific namespaces, override the default chart behaviour. Replace the default `ClusterRole` and `ClusterRoleBinding` in the target cluster with namespace-scoped `Role` and `RoleBinding` resources. This limits the Service Mirror's visibility (and the blast radius of a stolen token) to only the designated namespaces
- Because the Helm chart generates a static, long-lived Service Account token secret, it does not automatically expire. Implement a robust credential rotation pipeline to periodically invalidate and regenerate this cross-cluster token, limiting the viable exposure window if the secret is compromised
- Enforce strict Kubernetes RBAC within the source cluster, denying `get` and `list` operations for secrets in the `linkerd` and `linkerd-multicluster` namespaces to

all users, groups, and service accounts except the authorised Service Mirror controller itself

- Configure Kubernetes API audit logging on the target cluster and integrate it with a SIEM. Alert on any access patterns originating from the multicluster Service Account that deviate from standard controller reconciliation loops, such as rapid enumeration of all cluster namespaces

References:

- https://linkerd.io/2.20/features/multicluster/
- https://kubernetes.io/docs/reference/access-authn-authz/rbac/
- https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1552.007: Unsecured Credentials: Container API T1613: Container and Resource Discovery	AC-3: Access Enforcement AC-6: Least Privilege SC-28: Protection of Information at Rest SC-13: Cryptographic Protection

T-14 - Data exposure and compliance violations via inappropriate service mirroring or federation

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	Medium	Low	Linkerd Multicluster

Impact: Sensitive traffic can be routed to a lower-trust cluster, an incorrect tenant boundary, or a different geographical region. This may expose the data to weaker security controls or violate data residency and compliance regulations.

Description: Linkerd uses Kubernetes labels to dictate which services are shared across cluster boundaries. Standard service mirroring, determined via the `mirror.linkerd.io/exported=true` label in the hierarchical (gateway) mode and the `mirror.linkerd.io/exported=remote-discovery` label in peer-to-peer mode, creates a remote counterpart of a local service. Service federation, controlled via the `mirror.linkerd.io/federated=member` label, merges local and remote endpoints, allowing the mesh to distribute traffic across multiple clusters under a single service name.

If these labels are inappropriately applied to sensitive services, either through configuration error or malicious tampering by an attacker with RBAC access to modify `Service` resources, traffic routes across unintended boundaries. Data meant for a specific environment can be diverted to a mirrored or federated endpoint in a distinct trust zone, such as a different tenant's cluster in a hub-and-spoke architecture, a lower-trust environment, or another geographic region. This exposes payloads to interception and risks breaching data sovereignty regulations.

Recommendations:

- Restrict Kubernetes RBAC permissions for Linkerd `Link` CRs and `Service` resources
- Restrict the application of `mirror.linkerd.io/exported` and `mirror.linkerd.io/federated` labels to approved namespaces and workloads via admission controller policies (e.g. using Kyverno or OPA Gatekeeper)
- Establish strict governance and logical isolation boundaries to ensure federated services are only created between clusters sharing equivalent security postures, compliance baselines, and data-handling requirements
- Grant least privilege RBAC permissions to service mirror credentials, binding its Service Account to roles within specific, expected namespaces instead of granting cluster-wide read access
- Configure firewall rules to block communication between disparate trust zones

References:

- https://linkerd.io/2.20/reference/multicluster/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1562.001: Impair Defenses: Disable or Modify Tools T1195.002: Supply Chain Compromise: Compromise Software Supply Chain	SI-10: Information Input Validation SC-46: Cross Domain Policy Enforcement AC-3: Access Enforcement AC-6: Least Privilege

T-15 - Cross-cluster policy drift

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	Medium	Linkerd Multicluster Linkerd Policy Configuration Linkerd Policy Integrity

Impact: Inconsistent policy enforcement in multi-cluster architectures creates a fragmented zero-trust boundary. When equivalent services exhibit different security postures across clusters, it leads to unpredictable access control decisions and varying levels of exposure, depending on the cluster they reside in. In the case of federated services, the overarching security is degraded to its lowest common denominator, potentially leading to unpredictable access control decisions and unintended exposure of sensitive data.

Description: Because Linkerd operates a discrete control plane within each cluster rather than relying on a centralised control plane, it evaluates and enforces authorisation policies independently at the local level. When deploying equivalent services across multiple clusters, maintaining synchronised security postures is essential. If policy drift occurs, it creates exploitable discrepancies across the infrastructure.

For cross-cluster load-balanced services, traffic might be routed to a target cluster with more permissive policies, resulting in an intermittent bypass of security rules and degrading the application's overall security to its lowest common denominator.

For non-federated services, such as equivalent services in distinct regional spoke clusters, policy drift means an attacker compromising a workload in a poorly configured cluster can access local services that are correctly protected in other regions, undermining the consistency of the organisation's security baseline.

Recommendations:

- Manage Linkerd `Server` and `AuthorizationPolicy` resources through a centralised GitOps workflow (such as Flux CD or Argo CD). Deploying access controls from a single source of truth ensures consistent enforcement across all linked clusters and automatically reverts manual alterations
- Implement an automated policy auditing mechanism or utilise a policy engine (such as Kyverno or OPA Gatekeeper) to continuously monitor Linkerd custom resources across all clusters. Configure these tools to detect and alert on any configuration deviations or drift from the established baseline

References:

- https://linkerd.io/2.20/reference/multicluster/
- https://linkerd.io/2.20/reference/authorization-policy/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1562.001: Impair Defenses: Disable or Modify Tools T1195.002: Supply Chain Compromise: Compromise Software Supply Chain	CM-2: Baseline Configuration CM-3: Configuration Change Control

T-16 - Cluster compromise via Linkerd-Viz bundled Prometheus `nodes/proxy GET` permissions

Severity	CIA Impact			Affected Categories
	C	I	A	
High	High	High	High	Linkerd Viz RBAC

Impact: An attacker who compromises the bundled Linkerd-Viz Prometheus instance can abuse its overly permissive `ServiceAccount` token to interact directly with the Kubelet API on any node. This allows the attacker to execute arbitrary commands within any pod in the cluster, leading to total cluster compromise.

Description: The `linkerd-viz` extension bundles a Prometheus instance to scrape mesh metrics. By default, its corresponding `ClusterRole` grants `get`, `list`, and `watch` permissions on the `nodes` resource and the highly privileged `nodes/proxy` subresource. The `nodes/proxy` permission allows API server requests to be proxied directly to the Kubelet on any worker node. If an attacker extracts this token (e.g., via a vulnerability in Prometheus), they can bypass standard RBAC pod controls and invoke the Kubelet's `exec` API to achieve remote code execution (RCE) across the entire cluster.

Recommendations:

- Disable the bundled Prometheus instance by setting `prometheus.enabled: false` in the Helm chart if you already utilise a centralised monitoring stack. Provide a `prometheusUrl` pointing to your existing stack to eliminate the need for Linkerd to manage this high-privilege `ClusterRole`
- Remove the `nodes/proxy` subresource from the Prometheus `ClusterRole` if you must use the bundled instance, limiting its access to what is necessary
- Utilise fine-grained Kubelet API authorisation (supported in recent Kubernetes versions) to restrict the Prometheus Service Account to required read-only metric endpoints (like `/metrics/cadvisor`), rather than granting full proxy access

References:

- https://grahamhelton.com/blog/nodes-proxy-rce
- https://linkerd.io/2.20/features/dashboard/
- https://linkerd.io/2.20/tasks/external-prometheus/
- https://kubernetes.io/blog/2026/04/24/kubernetes-v1-36-fine-grained-kubelet-authorization-ga/

- https://kubernetes.io/docs/reference/access-authn-authz/kubelet-authn-authz/#fine-grained-authorization

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1528: Steal Application Access Token T1068: Exploitation for Privilege Escalation	AC-6: Least Privilege CM-7: Least Functionality

T-17 - Mesh-wide outage via trust anchor or identity issuer certificate expiration

Severity	CIA Impact			Affected Categories
	C	I	A	
High	Low	Low	High	Secrets and Key Management Linkerd Control Plane Default Configuration

Impact: Failure to rotate the Linkerd cryptographic materials may lead to a cluster-wide denial of service for all meshed applications. If the Identity Issuer certificate expires, the control plane cannot generate new short-lived workload certificates, causing a cascading, rolling outage over 24 hours as existing proxy certificates expire. If the root Trust Anchor expires, proxies immediately lose the ability to validate peer identities, resulting in an instantaneous halt of all inter-service mTLS traffic.

Description: Linkerd relies on a two-tier PKI architecture to enforce mTLS: a root **Trust Anchor** used by proxy sidecars to cryptographically validate the identities of their peers, and an intermediate **Identity Issuer** certificate used by the control plane's `identity` service to sign short-lived (typically 24-hour) mTLS leaf certificates for workloads. By default, the `linkerd install` CLI command generates both a trust anchor and a cluster issuer certificate that expire after 365 days.

If administrative processes fail to rotate these certificates before their respective expiration dates, the mesh fails, but the mechanisms differ:

- **Identity Issuer Expiration**: The `identity` service loses its ability to mint new workload certificates. Because existing leaf certificates remain valid until their TTL expires, the mesh does not fail instantly. Instead, as each sidecar proxy's short-lived certificate hits its 24-hour expiration, it fails to renew, leading to a cascading wave of connection rejections across the cluster
- **Trust Anchor Expiration:** The fundamental root of trust is broken. Even if individual proxy leaf certificates are still within their validity window, receiving proxies will immediately reject incoming connections because they can no longer validate the certificate chain against a valid root. This causes a sudden, total disruption of all meshed communication

Furthermore, because multicluster communication requires a shared trust anchor, the default `linkerd install` setup is incompatible with multi-cluster environments, requiring explicit trust anchor provisioning from the start.

Recommendations:

- Explicitly provision and pass cryptographic material during Linkerd installation rather than relying on auto-generated defaults. Store the trust anchor's private key securely out-of-cluster in a dedicated Key Management Service (KMS) or HSM. Only deploy the public trust bundle and the intermediate issuer certificate and key to the cluster, passing them explicitly into your deployment configuration (e.g., via Helm values)
- Automate the lifecycle and rotation of the Linkerd issuer certificates using `cert-manager` (or a similar robust PKI automation tool) integrated with the cluster, removing the reliance on manual administrative intervention
- Plan and document the operational procedure for rotating the root trust anchor. Because this is a highly disruptive, non-trivial task, it must be executed as a phased rollout:
    - Deploy a trust bundle containing both the old and new anchors
    - Sequentially restart the control plane and meshed workloads to trust both anchors
    - Rotate the identity issuer to a new certificate signed by the *new* trust anchor, and restart workloads again to fetch leaf certificates chaining to the new anchor
    - Finally, remove the old anchor from the trust bundle and perform a final restart
- Utilise monitoring and external telemetry systems to proactively track the expiration dates of both the trust anchor and issuer certificates, configuring alerts well in advance of their expiration
- Develop an incident response runbook for expired certificate recovery

References:

- https://linkerd.io/2.20/features/automatic-mtls/#operational-concerns
- https://linkerd.io/2.20/tasks/automatically-rotating-control-plane-tls-credentials/
- https://linkerd.io/2.20/tasks/manually-rotating-control-plane-tls-credentials/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1499: Endpoint Denial of Service	SC-12: Cryptographic Key Establishment and Management SI-4: System Monitoring CP-2: Contingency Plan

T-18 - Linkerd control plane denial of service

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Low	Low	High	Linkerd Control Plane Network

Impact: Availability disruption of any Linkerd control plane component degrades the operational state of the service mesh. Depending on which specific service is overwhelmed, new mTLS certificates may not be issued (eventually causing traffic drops), proxies may stop receiving policy updates, or new pods may either be rejected at admission or be scheduled unmeshed (depending on the proxy injector's webhook failure policy).

Description: By default, Linkerd installs its core control plane services (such as the `destination`, `identity`, or `proxy-injector` controllers) as single-replica deployments. This configuration leaves each individual service vulnerable as a structural single point of failure. An attacker, a compromised workload, or an internal misconfiguration (such as a crash-looping deployment continuously triggering webhook requests) can target a specific component by flooding it with excessive API calls or gRPC connections.

Under such load, the targeted service can suffer from resource exhaustion and crash, breaking the specific mesh operations it provides. The exact consequence depends on the targeted component: an offline `identity` service prevents the issuance and rotation of workload mTLS certificates (eventually causing cross-pod communication to fail as existing certificates expire), an offline `destination` service halts policy updates, while an overwhelmed `proxy-injector` will either prevent new workloads from being scheduled (if configured securely) or allow them to be deployed without a sidecar.

Recommendations:

- Deploy the Linkerd control plane in High Availability (HA) mode for all production environments. This provisions multiple replicas of each component backed by Kubernetes Pod Anti-Affinity rules, eliminating single points of failure and distributing the load across separate physical worker nodes
- Configure `PodDisruptionBudgets` (PDBs) for all Linkerd control plane deployments. When running in HA mode, PDBs ensure that Kubernetes does not voluntarily evict too many replicas simultaneously during node maintenance or autoscaling events, guaranteeing a minimum quorum of management services remains online to handle load
- Ensure the `proxy-injector MutatingAdmissionWebhook` is configured with a `failurePolicy: Fail`. This guarantees that if the injector is unreachable during a

DoS event, pods will fail to schedule rather than failing open and running without a proxy sidecar (sacrificing availability to preserve zero-trust integrity)

- Assign Kubernetes `PriorityClass` values to all Linkerd control plane pods. This ensures that in the event of node-level resource contention, the Kubernetes scheduler prioritises keeping these individual management services online over less critical application workloads
- Enforce CPU and memory requests and limits for all control plane deployments to prevent a localised traffic spike against one targeted component from consuming all available node resources and starving the rest of the system
- Implement Kubernetes `NetworkPolicies` to restrict inbound traffic to the control plane namespace. While mesh proxies must communicate with the `identity` and `destination` components, access to the `proxy-injector` webhook should be restricted exclusively to the Kubernetes API server

References:

- https://linkerd.io/2.20/features/ha/
- https://linkerd.io/2.20/reference/architecture/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1499: Endpoint Denial of Service	CM-6: Configuration Settings SC-5: Denial-of-service Protection SC-6: Resource Availability

T-19 - Mesh management outage via webhook TLS certificate expiration

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Low	Low	High	Admission Control Default Configuration

Impact: Upon expiration of the webhook TLS certificates, the Kubernetes API server will reject communication with Linkerd's admission webhooks. This halts the scheduling of new meshed pods (as sidecar injection fails) and blocks administrators from applying or modifying Linkerd routing and authorisation policies.

Description: The Linkerd control plane relies on Kubernetes Mutating and Validating Admission Webhooks (specifically the `proxy-injector`, `sp-validator`, and `policy-validator`) to inject proxy sidecars into workloads and validate custom resource configurations. Kubernetes requires that all communication with admission webhooks be served over TLS.

By default, if an external certificate is not explicitly provided during installation, the Linkerd Helm chart automatically generates self-signed certificates with a hardcoded expiration of 365 days for these webhooks. Because it is generated locally at Helm render-time, it lacks any native mechanism for automatic rotation.

One year after the initial installation, these certificates will expire. When the Kubernetes API server subsequently attempts to call the webhooks for pod admission or policy validation, the TLS handshake will fail, with consequences depending on each webhook's configured `failurePolicy`.

For the `proxy-injector`, a failure will either block the creation of new workloads across the cluster or allow them to spawn unmeshed, bypassing the mesh security controls. For the `policy-validator` and `sp-validator`, TLS failures will either prevent operators from creating or updating authorisation policies and routing configurations, or allow the admission of invalid custom resources. Admitting malformed configurations can result in runtime routing disruptions or unintended access control vulnerabilities.

Recommendations:

- Never rely on the default, Helm-generated self-signed webhook certificates for long-lived or production environments
- Configure the Linkerd Helm values to manage webhook certificates externally by setting `externalSecret: true` for the relevant control plane components

- Use a PKI automation tool like `cert-manager` to automatically provision, inject, and proactively rotate webhook TLS credentials

References:

- https://linkerd.io/2.20/tasks/automatically-rotating-webhook-tls-credentials/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1499: Endpoint Denial of Service	SC-12: Cryptographic Key Establishment and Management CP-2: Contingency Plan

T-20 - Cross-cluster traffic disruption via out-of-sync trust anchor rotation

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Low	Low	Medium	Linkerd Multicluster Secrets and Key Management

Impact: Desynchronised trust anchors across linked multicluster environments lead to cross-cluster mTLS validation failures. This disruption may pressure operators into temporarily disabling strict mTLS or weakening authorisation policies to restore connectivity, degrading the overall cluster security posture.

Description: Linkerd's multicluster architecture requires all participating clusters to share a common trust anchor (root certificate) to cryptographically validate cross-cluster mTLS traffic. When this shared trust anchor requires rotation, it must be carefully managed across all clusters using a bundled approach. If the rotation process becomes desynchronised—for example, if one cluster begins issuing workload certificates chained to a new root while a remote cluster's proxies still exclusively trust the old root—cross-cluster connections will be rejected. This results in partial or intermittent traffic drops for mirrored services that persist until the updated trust bundle is universally distributed and recognised by all sidecars and gateways across the multicluster topology.

Recommendations:

- Implement a well documented, GitOps-driven procedure for multicluster trust anchor rotation that relies on trust bundle appending. Always distribute a combined bundle containing both the old and new root certificates to all clusters, and restart all proxies to recognise the bundle, before rotating the active intermediate issuer in any individual cluster

References:

- https://linkerd.io/2.20/features/multicluster/
- https://linkerd.io/2.20/tasks/installing-multicluster/#trust-anchor-bundle

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1499: Endpoint Denial of Service	SC-12: Cryptographic Key Establishment and Management

T-21 - Multicluster gateway overexposure and denial of service

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Low	Low	High	Linkerd Multicluster - Hierarchical Mode Network Default Configuration

Impact: An attacker can target the exposed `LoadBalancer` IP of a multicluster gateway with volumetric or connection-exhaustion attacks, causing cross-cluster traffic disruptions.

Description: The Linkerd hierarchical multicluster deployment mode relies on exposing a `linkerd-gateway` component, typically via a Kubernetes `LoadBalancer` service, to receive inbound traffic from remote clusters. If the gateway's external IP or exposed port (typically 4143) is left open to the internet, it becomes a highly visible target. While unauthorised users cannot easily bypass the mTLS authentication required to successfully route payloads through the gateway, an attacker can still initiate a DoS attack by flooding the endpoint with TCP connection requests, SYN floods, or expensive TLS handshake initiations. This resource exhaustion can crash the gateway pods, breaking cross-cluster communication.

Recommendations:

- Restrict network access to the multicluster gateway's `LoadBalancer` IP at the cloud provider infrastructure level. Configure cloud firewall rules to allow ingress traffic *only* from the known CIDR blocks or static NAT/egress IP addresses of trusted remote clusters
- Deploy the `linkerd-multicluster` extension in its High Availability (HA) mode using the Helm chart to automatically provision the `linkerd-gateway` with multiple replicas and Pod Anti-Affinity rules
- Configure strict CPU and memory requests and limits for the gateway proxy and deployment via Helm overrides. This ensures an anomalous connection flood cannot cause node-level resource exhaustion
- Apply rate limiting and connection tracking at the cloud load balancer or edge firewall layer to drop anomalous connection floods before they reach the Kubernetes cluster and the `linkerd-gateway` proxy
- Implement monitoring and alerting for gateway resource utilisation, specifically tracking Linkerd metrics for connection drops, memory saturation, and CPU throttling on the gateway pods

References:

- https://linkerd.io/2.20/features/multicluster/#hierarchical-networks
- https://github.com/linkerd/linkerd2/blob/main/multicluster/charts/linkerd-multicluster/values-ha.yaml

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1499: Endpoint Denial of Service T1498: Network Denial of Service	SC-5: Denial-of-service Protection SC-7: Boundary Protection SI-4: System Monitoring

T-22 - Residual access via incomplete multicluster unlinking

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Medium	Medium	Low	Linkerd Multicluster

Impact: Incomplete removal of a multicluster link leaves behind stale credentials, mirrored services, and open network pathways. A compromised former peer cluster could exploit these remnants to regain unauthorised access to the mesh.

Description: Unlinking Linkerd clusters often requires more than just deleting the `Link` CR. Artifacts such as mirrored Kubernetes `Services`, multicluster gateways, `ServiceAccount` credentials, and cloud firewall rules may persist and potentially leave latent access paths open. An attacker in the previously linked remote cluster could leverage such forgotten credentials and open boundaries to enumerate local services or inject unauthorised traffic.

Recommendations:

- Maintain a strict multicluster offboarding runbook to ensure consistent and complete teardowns
- Manually audit and delete any lingering mirrored services (e.g., services suffixed with the remote cluster's name) to prevent traffic routing to stale endpoints
- Update cloud provider firewall rules to revoke ingress access to the `linkerd-gateway` `LoadBalancer` IP from the unlinked cluster

References:

- https://linkerd.io/2.20/tasks/uninstall-multicluster/#unlinking

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1078: Valid Accounts T1133: External Remote Services T1021: Remote Services	CM-1: Policy and Procedures CM-3: Configuration Change Control

T-23 - Workload identity spoofing via proxy credential theft

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Medium	Medium	Low	Linkerd Data Plane Kubernetes

Impact: An attacker who compromises a Linkerd sidecar proxy can extract its mTLS private key and certificate. This allows them to impersonate the compromised workload from elsewhere on the network for the remaining duration of the certificate's lifetime.

Description: Linkerd enforces zero-trust security by issuing mTLS certificates to proxy sidecars, cryptographically proving a pod's identity. If an attacker gains sufficient privilege to dump the proxy's process memory, they can extract the active keypair. Once exported, these credentials can be used to bypass `AuthorizationPolicy` rules. The attacker maintains access only for the remaining lifespan of that specific certificate (typically less than 24 hours).

Recommendations:

- Ensure that the Linkerd control plane is configured to issue workload certificates with short lifetimes, not exceeding the secure default 24 hours, to minimise the window of opportunity for abusing an extracted credential
- Enforce a baseline of container hardening, such as the Kubernetes Restricted Pod Security Standard (PSS). This requires setting `allowPrivilegeEscalation: false` and dropping all capabilities (including `CAP_SYS_PTRACE`, which is required to dump process memory). Additionally, use a read-only root filesystem to make post-exploitation activities more difficult
- Implement network-level observability and SIEM alerting to monitor for anomalous certificate usage, such as a specific Linkerd identity being presented from an unexpected, unassigned IP address or originating from outside the known cluster network boundaries
- Implement a runtime security tool, such as Falco or Tetragon, to monitor container behaviour at the kernel level. Configure alerts for anomalous activity, such as unexpected processes attempting to read sensitive paths (such as certificate private keys) or using `ptrace` system calls

References:

- https://linkerd.io/2.20/features/automatic-mtls/
- https://kubernetes.io/docs/concepts/security/pod-security-standards/
- https://github.com/falcosecurity/falco
- https://tetragon.io/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1212: Exploitation for Credential Access T1055.008: Process Injection: Ptrace System Calls T1649: Steal or Forge Authentication Certificates	CM-2: Baseline Configuration CM-6: Configuration Settings CM-7: Least Functionality SC-39: Process Isolation SI-3: Malicious Code Protection SI-4: System Monitoring

T-24 - L7 policy evasion via protocol detection bypass or opaque port misconfiguration

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Medium	Medium	Medium	Linkerd Data Plane Linkerd Policy Configuration

Impact: Attackers can bypass fine-grained L7 access controls (such as path-based `HTTPRoute` policies) by forcing the proxy to treat structured traffic as an uninspected TCP stream. Additionally, misconfigured opaque ports can silently disable L7 security for HTTP services or cause severe 10-second latency penalties for legitimate server-first protocols.

Description: Linkerd dynamically identifies application protocols by sniffing the first few bytes of a connection. If it cannot determine the protocol within 10 seconds, or if the destination port is marked as `opaque`, the proxy defaults to treating the connection as a raw TCP stream. Attackers may exploit this fallback mechanism by deliberately trickling data to hit the 10-second timeout, or potentially by spoofing Application-Layer Protocol Negotiation (ALPN) indicators, causing the proxy to bypass HTTP or gRPC-specific authorisation rules. Furthermore, operational risks arise if developers inadvertently deploy HTTP services on Linkerd's default opaque ports (e.g., 3306, 5432), which disables L7 observability and security, or if they fail to mark genuine server-first protocols as `opaque`, subjecting legitimate traffic to mandatory 10-second delays.

Recommendations:

- Explicitly define the `proxyProtocol` (e.g., `HTTP/1`, `HTTP/2`, `gRPC`) in Linkerd `Server` resources, or the `appProtocol` for standard Kubernetes `Service` resources, for all workloads. This disables protocol sniffing and drops non-conforming traffic, neutralising timeout evasion tactics
- Audit application configurations against Linkerd's `default-opaque-ports` list to ensure no HTTP or gRPC services are inadvertently bypassing L7 inspection
- Annotate custom, non-standard ports that handle server-first or non-HTTP protocols with `config.linkerd.io/opaque-ports` to prevent the proxy from hanging and incurring the 10-second timeout penalty

References:

- https://linkerd.io/2.20/features/protocol-detection/
- https://linkerd.io/2026/02/09/linkerd-protocol-detection/
- https://linkerd.io/2021/02/23/protocol-detection-and-opaque-ports-in-linkerd/
- https://linkerd.io/2.20/common-errors/protocol-detection/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1001.003: Data Obfuscation: Protocol or Service Impersonation T1571: Non-Standard Port	CM-7: Least Functionality

T-25 - L7 policy evasion via proxy-application parser differentials

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Medium	Medium	Low	Linkerd Data Plane Linkerd Policy Configuration

Impact: Attackers can exploit the semantic gap between how the Linkerd proxy and the application backend interpret ambiguous HTTP requests to bypass fine-grained L7 policies. This allows unauthorised access to restricted endpoints, as the proxy may permit a seemingly benign request that the backend subsequently routes to a protected resource.

Description: Linkerd enforces L7 policies based on the proxy's interpretation of HTTP requests, creating a potential vulnerability if the backend application framework parses ambiguous URIs, paths, or headers differently. Attackers may exploit such semantic gaps and parser differentials—such as discrepancies in path normalisation, case sensitivity, duplicate headers, URL encoding, or techniques like HTTP request smuggling or desync attacks—to trick the proxy into permitting malicious traffic that the backend subsequently routes to protected resources.

Notably, Linkerd lacks configurable proxy-level path normalisation settings, placing the burden of mitigating URI ambiguity on the backend application and the exactness of route definitions.

Recommendations:

- Ensure backend application frameworks are configured to perform strict path normalisation and proactively reject ambiguous or malformed requests (e.g., returning a `400 Bad Request`) before routing them to application logic
- Design Linkerd `HTTPRoute` definitions to be as explicit and restrictive as possible, accounting for the specific backend framework's routing behaviours, such as how it handles trailing slashes or case insensitivity
- Implement defence-in-depth by enforcing application-layer authorisation for sensitive services so that security does not rely entirely on the proxy's initial URI evaluation
- Conduct regular application security testing, including fuzzing assessments, to identify and remediate parsing discrepancies between the Linkerd proxy and the specific backend stacks in use

References:

- https://linkerd.io/2.20/reference/httproute/
- https://iterasec.com/blog/understanding-parser-differential-vulnerabilities/
- https://about.gitlab.com/blog/how-to-exploit-parser-differentials/
- https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1001.003: Data Obfuscation: Protocol or Service Impersonation T1210: Exploitation of Remote Services	SI-10: Information Input Validation

T-26 - Overly permissive or absent `EgressNetwork` policies

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Medium	Low	Low	Linkerd Policy Configuration Network

Impact: Overly permissive or absent Linkerd `EgressNetwork` policies can allow compromised workloads can establish unrestricted outbound network connections to the internet, allowing attackers to exfiltrate sensitive data, download secondary malicious payloads, or establish communication with external command-and-control (C2) servers.

Description: By default, workloads within a Kubernetes cluster often possess unrestricted access to external networks unless denied by L3/L4 Kubernetes Network Policies. Linkerd utilises `EgressNetwork` CRs to govern and restrict outbound traffic from the mesh to external destinations. If these Linkerd policies are missing, or if they are configured with overly broad CIDR blocks (such as `0.0.0.0/0`), the mesh fails to provide egress boundary protection. Consequently, in the absence of underlying network-level isolation, a malicious actor who compromises a meshed pod can route traffic outside the cluster boundaries.

Notably, the official Linkerd documentation warns that a service mesh cannot provide strong security guarantees regarding egress traffic on its own. Because an attacker with sufficient pod-level privileges could potentially bypass the proxy sidecar (T-08), thereby evading Linkerd's egress controls, fully restricting outbound traffic requires a defence-in-depth approach. Relying solely on the mesh is insufficient; `EgressNetwork` configurations must be paired with robust Kubernetes Network Policies or external firewall rules to ensure comprehensive egress security.

Recommendations:

- Implement default-deny Kubernetes egress Network Policies as a foundational defence-in-depth measure, ensuring that network-level controls block outbound traffic even if mesh policies are misconfigured
- Define explicit Linkerd `EgressNetwork` CRs that restrict outbound traffic to the approved, specific external IP ranges required for the application to function
- Utilise external DNS-based egress controls or route outbound traffic through a dedicated egress gateway or firewall to enable domain-level filtering and deep packet inspection
- Continuously monitor Linkerd proxy telemetry and cluster network logs for anomalous outbound connection attempts, configuring alerts for traffic destined for unrecognised IP spaces

References:

- https://linkerd.io/2.20/features/egress/
- https://linkerd.io/2.20/reference/egress-network/
- https://kubernetes.io/docs/concepts/services-networking/network-policies/
- https://docs.aws.amazon.com/network-firewall/latest/developerguide/how-does-network-firewall-proxy-work.html

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1105: Ingress Tool Transfer T1567: Exfiltration Over Web Service T1041: Exfiltration Over C2 Channel	AC-4: Information Flow Enforcement SC-7: Boundary Protection

T-27 - Plaintext data interception via local loopback snooping

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	Medium	Low	Low	Linkerd Data Plane Kubernetes

Impact: Attackers who have already breached a specific application container or its underlying node can bypass the mesh's mTLS encryption to passively intercept sensitive plaintext data destined for, or originating from, that specific pod.

Description: While mTLS encrypts data in transit across the cluster network, the sidecar service mesh architecture inherently necessitates a plaintext boundary within the pod itself. Communication between the application container and its associated Linkerd micro-proxy occurs over the pod's local loopback network interface (`localhost` or `lo`). An attacker who manages to achieve arbitrary code execution within the application container, or compromises the underlying Kubernetes worker node and enters the pod's network namespace, can utilise standard packet sniffing utilities (such as `tcpdump`) to intercept loopback plaintext traffic.

Recommendations:

- Enforce read-only root filesystems (`readOnlyRootFilesystem: true`) for all application containers to restrict an attacker's ability to download packet capture utilities post-compromise
- Enforce the `Restricted` Pod Security Standards (PSS) profile at the namespace level, explicitly dropping all unnecessary Linux capabilities (`securityContext.capabilities.drop: ["ALL"]`). Specifically, denying the `NET_RAW` capability is critical, as it prevents even statically compiled, 'dropped-in' binaries from opening the raw sockets required to sniff network traffic on the loopback interface
- Apply strict `seccomp` profiles to the application workloads to filter out the system calls required for network diagnostics and raw socket creation, further hardening the container boundary against post-exploitation activities
- Harden the underlying Kubernetes worker nodes and restrict privileged node-level access to prevent internal actors or attackers from leveraging host access to enter application network namespaces

References:

- https://linkerd.io/2.20/features/automatic-mtls/

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1105: Ingress Tool Transfer T1040: Network Sniffing	CM-6: Configuration Settings CM-7: Least Functionality SC-3: Security Function Isolation SC-39: Process Isolation

T-28 - Linkerd-Viz Tap API abuse for real-time traffic interception

Severity	CIA Impact			Affected Categories
	C	I	A	
Medium	High	Low	Low	Linkerd Viz RBAC

Impact: Unauthorised access to the tap API allows attackers to intercept live inter-service traffic, exposing sensitive data in transit (such as authentication tokens and confidential HTTP headers).

Description: The `linkerd-viz` extension's `tap` API allows for dynamic introspection of live traffic between meshed pods. If this API is left unsecured due to overly permissive Kubernetes RBAC or unrestricted network access, an attacker with a compromised `ServiceAccount` can silently wiretap communications. This enables them to capture real-time request and response headers, turning a valuable debugging tool into a data exfiltration vector.

Recommendations:

- Restrict Kubernetes RBAC for the `tap.linkerd.io` API group (e.g., the `linkerd-viz-tap-admin ClusterRole`) to authorised administrative Service Accounts
- Install Linkerd Viz with `dashboard.restrictPrivileges=true` to revoke the web dashboard's default cluster-wide `tap` privileges
- Configure `tap.ignoreHeaders` in the `linkerd-viz` Helm chart to redact sensitive HTTP headers (e.g. `Authorization`) from `tap` output
- Implement `AuthorizationPolicy` resources within the `linkerd-viz` namespace to limit network ingress to the tap server
- Consider disabling the `linkerd-viz` extension in highly sensitive production environments, relying instead on sanitised, out-of-band metrics and distributed tracing

References:

- https://linkerd.io/2.20/tasks/securing-linkerd-tap/
- https://linkerd.io/docs/reference/cli/viz/#tap

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1040: Network Sniffing	AC-3: Access Enforcement AC-6: Least Privilege AU-9: Protection of Audit Information CM-7: Least Functionality

T-29 - Denial of service and infrastructure overwhelm via telemetry resource exhaustion (CVE-2025-43915)

Severity	CIA Impact			Affected Categories
	C	I	A	
Low	Low	Low	Medium	Linkerd Data Plane

Impact: An attacker can intentionally trigger an explosion in metric cardinality by supplying a vast number of unique HTTP hostnames. This causes the Linkerd proxy to consume excessive memory leading to OOM crashes, overwhelms the cluster's metrics ingestion infrastructure, and can generate significant, unexpected financial costs for third-party metrics platforms.

Description: Linkerd proxies track and expose Prometheus metrics for a workload's inbound and outbound HTTP requests. Vulnerable versions of Linkerd include an `authority` label for inbound request metrics and a `hostname` label for outbound request metrics. If a proxy receives requests with an unbounded number of unique hostnames or authorities, it results in a massive increase in the cardinality of the generated metrics. Attackers can exploit this by crafting malicious URLs or HTTP `Host` / `:authority` headers and sending them to meshed workloads exposed to the internet (e.g., via meshed ingress controllers) or by compromising workloads that make arbitrary egress calls. In recent patched versions (`edge-25.2.1` and higher), **the maintainers disabled hostname metric labels by default, making them an opt-in feature instead via `proxy.metrics.hostnameLabels` (`false` by default), significantly reducing the risk unless explicitly re-enabled by an administrator**.

Recommendations:

- Ensure that `proxy.metrics.hostnameLabels` remain disabled (`false`) in the Linkerd control plane configuration, relying on Linkerd's secure default behaviour
- For unpatched deployments or deployments where these labels are necessary, filter incoming HTTP requests at the cluster edge (e.g., within the ingress controller or a Web Application Firewall) to drop requests with unrecognised, arbitrary, or unbound hostnames before they reach the Linkerd proxies
- Audit meshed workloads that make egress calls to untrusted domains, ensuring the number of unique target hostnames is bounded or that egress metrics are disabled for workloads acting as open forward proxies
- Enforce strict memory resource requests and limits for all Linkerd proxy sidecar containers to ensure that a cardinality explosion results in a localised OOM kill rather than node-level resource starvation
- Configure Prometheus scrape limits (`sample_limit` or `series_limit`) and apply metric relabelling rules at the ingestion layer to drop high-cardinality time series before they overwhelm the storage backend or inflate third-party billing

References:

- https://nvd.nist.gov/vuln/detail/CVE-2025-43915
- https://docs.buoyant.io/security/advisories/2025-01/
- https://cwe.mitre.org/data/definitions/770.html

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1499: Endpoint Denial of Service	CM-6: Configuration Settings CM-7: Least Functionality SC-5: Denial-of-service Protection

T-30 - Pod denial of service via proxy shutdown endpoint

Severity	CIA Impact			Affected Categories
	C	I	A	
Low	Low	Low	Medium	Linkerd Data Plane

Impact: An attacker who compromises an application container or exploits a Server-Side Request Forgery (SSRF) vulnerability can prematurely terminate the Linkerd proxy sidecar. This severs the pod's network connectivity, resulting in a localised denial of service and disrupting the application's availability.

Description: Historically, the Linkerd proxy exposed a local administrative endpoint (`http://localhost:4191/shutdown`) by default to allow application containers, particularly transient tasks like Kubernetes `Jobs` and `CronJobs`, to cleanly terminate the sidecar upon completion. If this endpoint is explicitly enabled (**it has been disabled by default since Linkerd 2.16**), it introduces a local attack vector. An attacker who achieves Remote Code Execution (RCE) within the application container, or exploits an SSRF vulnerability (CVE-2024-40632), can issue an unauthenticated `POST` request to this loopback endpoint. This shuts down the proxy, dropping all active connections and rendering the application container isolated from the mesh and the wider network.

Recommendations:

- Ensure the proxy shutdown endpoint remains disabled (`proxy.enableShutdownEndpoint: false`) in the Linkerd configuration, preserving the secure default behaviour of modern releases
- Leverage Kubernetes native sidecar containers (supported and enabled by default from Linkerd 2.20 onwards). This feature natively manages the lifecycle of proxy containers, ensuring they automatically shut down when the primary `Job` or `CronJob` workload terminates, eliminating the operational need to expose the administrative shutdown endpoint

References:

- https://linkerd.io/2.20/tasks/graceful-shutdown/
- https://linkerd.io/2.20/features/native-sidecars/
- https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/
- https://nvd.nist.gov/vuln/detail/CVE-2024-40632

Framework Mappings:

MITRE ATT&CK Mapping	NIST 800-53 Revision 5 Mapping
T1489: Service Stop T1499: Endpoint Denial of Service	CM-6: Configuration Settings CM-7: Least Functionality SC-5: Denial-of-service Protection

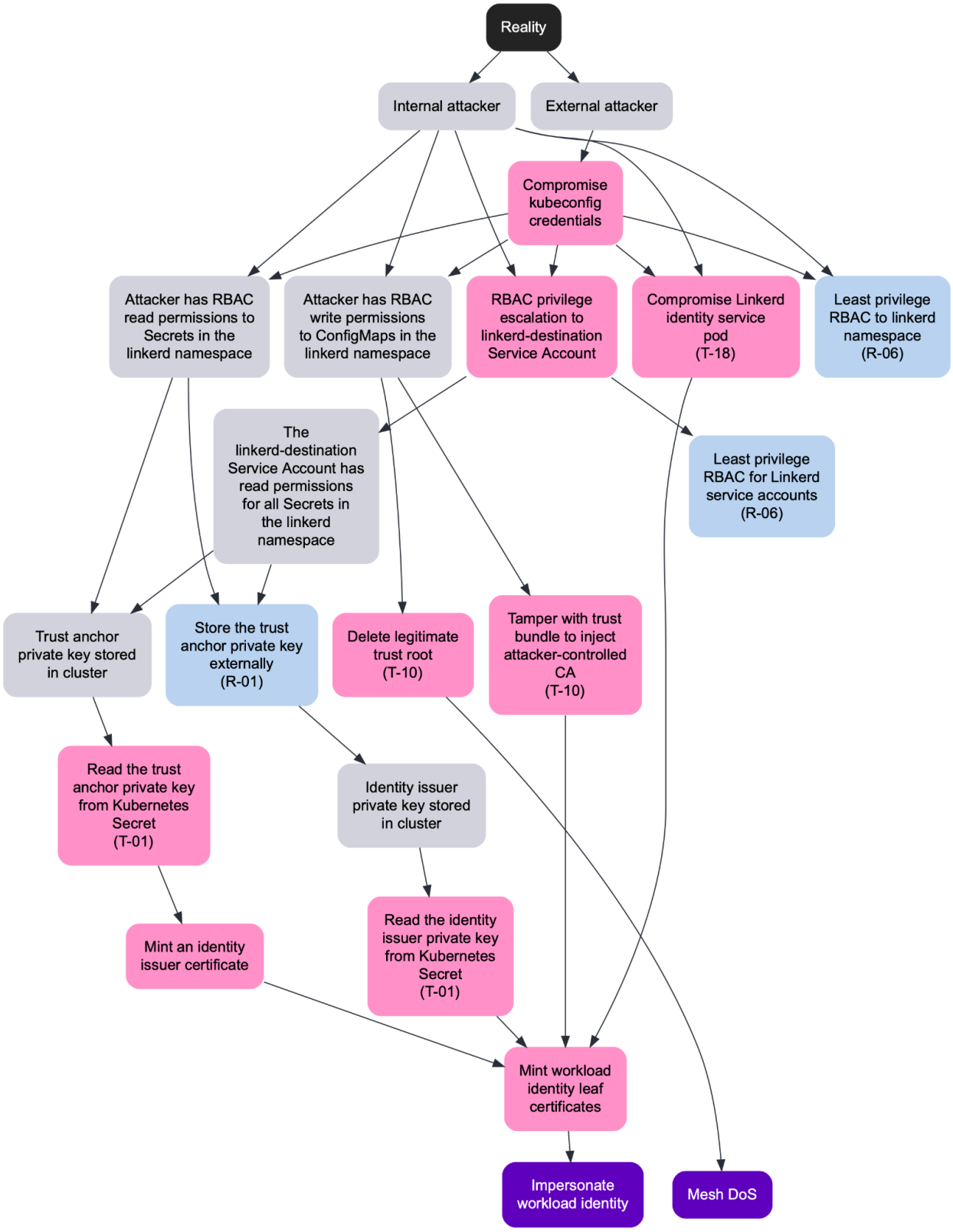
Attack trees

Attack trees have been developed with the Deciduous web tool, and should be reviewed with a top-down approach, from the starting point for each scenario (e.g., “Reality”) to the goal of the attack. Typically, each attack has multiple branches and multiple goals, all starting from the same assumptions. Attack paths include mitigations whenever a control is already in place to remediate an attack step.

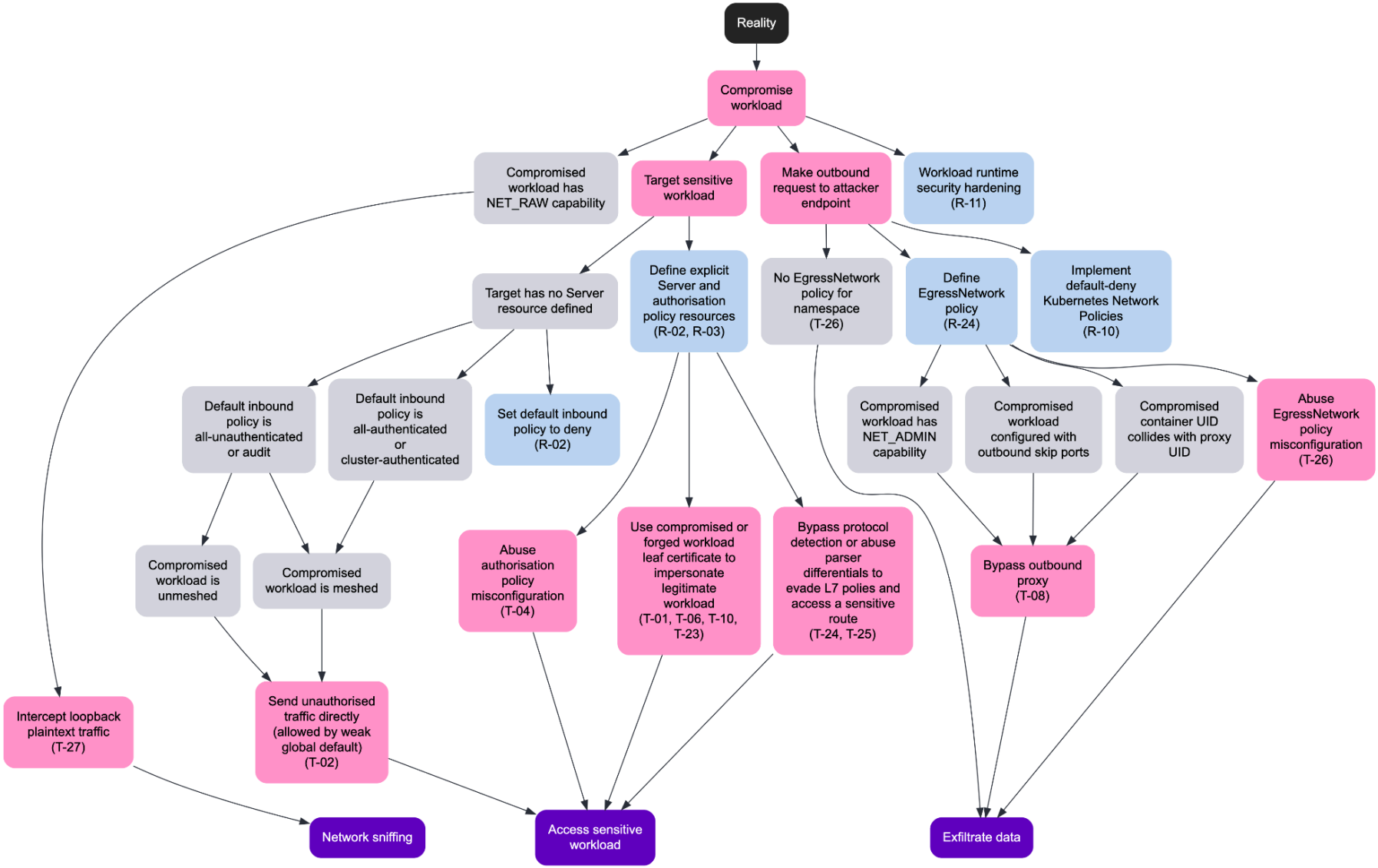
Attack tree node legend

Reality	The starting point for the scenario
Fact	Something that is true to the system, and it’s neither an attack or a mitigation
Attack	A step in an attack path
Mitigation	An existing control to mitigate an attack step
Goal	The end goal for the attack

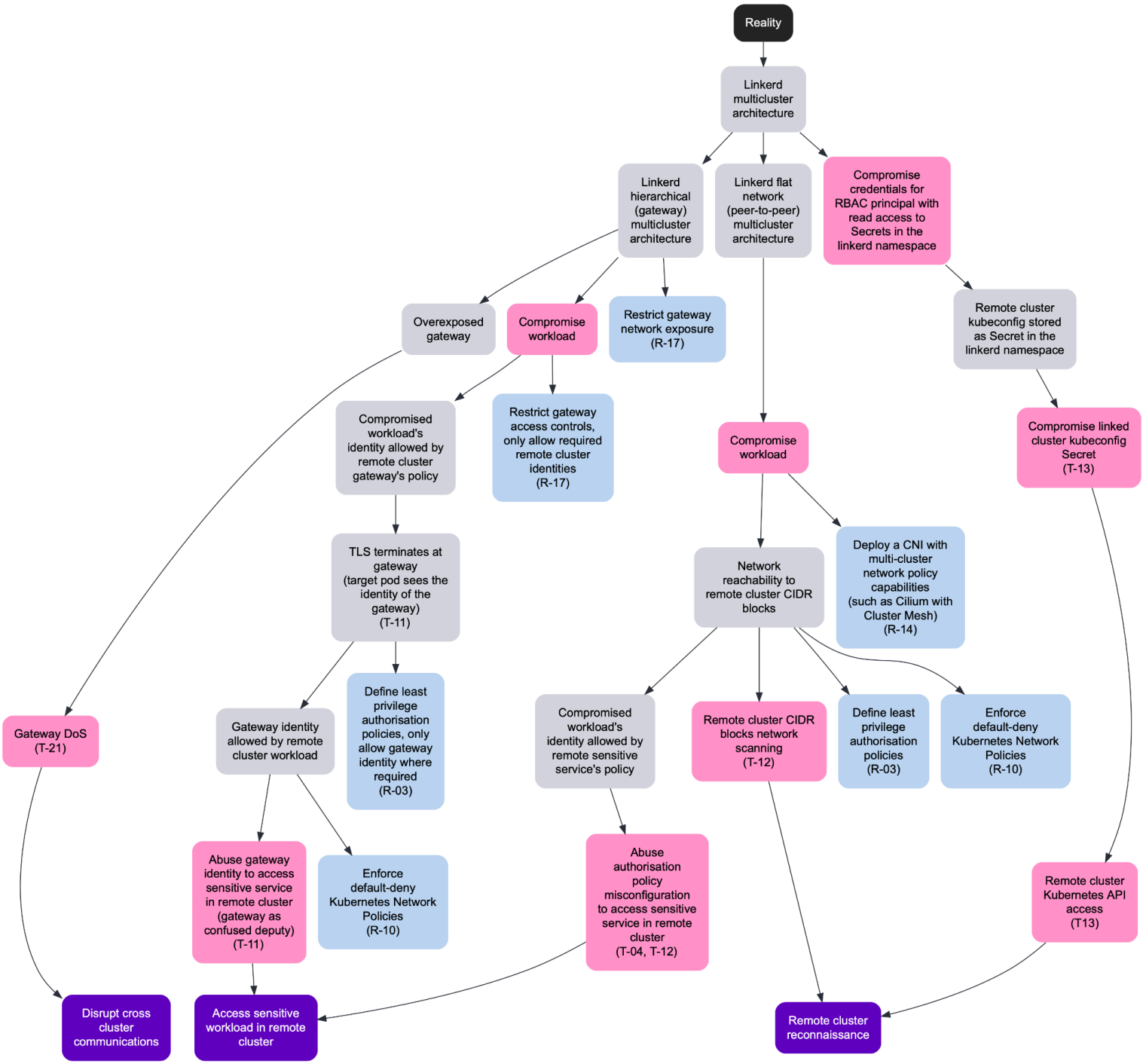
Mesh PKI compromise



Compromised workload - confidentiality threats



Multicluster threats



Hardening

Based on the preceding threat modeling exercise, a set of hardening recommendations is provided below. Recommendations cover several aspects, including the installation and custom resource configurations for Linkerd and related components.

Priority

Each recommendation has been evaluated using the following scoring system:

Red	High
Amber	Medium
Green	Low

Recommendations

ID: Title	Priority	Mitigated Threats	Description	NIST 800-53 Revision 5 Mapping
R-01: Store the Trust Anchor Private Key Externally	High	T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys	• Generate and store the mTLS trust anchor private key in a secure, offline environment or an external Key Management Service (KMS) rather than within the Kubernetes cluster	SC-12: Cryptographic Key Establishment and Management SC-17: Public Key Infrastructure Certificates SC-28: Protection of Information at Rest
R-02: Default Inbound Policy Hardening	High	T-02 - Unauthorised access to meshed services due to permissive default inbound policy	• Set a restrictive cluster-wide default inbound policy instead of the default `all-unauthenticated` • Optimally, set `proxy.defaultInboundPolicy` to `deny`, which blocks all incoming traffic by default and requires explicitly defined `Server` and `AuthorizationPolicy` resources for expected communication • Alternatively, or as a transitional step during policy rollout, set the default to `cluster-authenticated` or `all-authenticated`	AC-4: Information Flow Enforcement CM-6: Configuration Settings SC-8: Transmission Confidentiality and Integrity SC-23: Session Authenticity

R-03: Explicitly Define Least Privilege Authorisation Policy Resources	High	T-02 - Unauthorised access to meshed services due to permissive default inbound policy T-04 - Overly permissive or absent authorisation policies	• Create explicit `AuthorizationPolicy` resources for all workloads, linked to specific `Server` definitions and granular L7 `HTTPRoute` or `GRPCRoute` resources, to enforce strict zero-trust boundaries • Configure `MeshTLSAuthentication` to restrict allowed client mTLS identities, and `NetworkAuthentication` to restrict permitted source IP ranges	AC-3: Access Enforcement AC-4: Information Flow Enforcement SC-23: Session Authenticity
R-04: Proxy Injector Webhook Failure Policy Hardening	High	T-03 - Unmeshed workload admission due to permissive proxy injector webhook failure policy	• Ensure the Linkerd proxy injector webhook's `failurePolicy` is set to `Fail` instead of `Ignore` (`webhookFailurePolicy: Fail` in the Helm chart) • Deploy Linkerd with multiple replicas per controller to reduce the likelihood of admission controller unavailability triggering the bypass	CM-2: Baseline Configuration CM-6: Configuration Settings SC-7: Boundary Protection SC-8: Transmission Confidentiality and Integrity
R-05: Restrict Linkerd Data Plane Annotations via Admission Controller	High	T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation	• Deploy validating admission controllers (such as Kyverno or OPA Gatekeeper) to govern the use of pod or namespace annotations that	SI-10: Information Input Validation

			determine Linkerd proxy configuration or injection	
R-06: Least Privilege RBAC to Linkerd Namespaces	High	T-09 - Linkerd control plane compromise T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys T-10 - Identity spoofing or denial of service via trust bundle tampering T-13 - Remote cluster Kubernetes API access via link credentials theft	• Restrict Kubernetes RBAC permissions across the `linkerd` , `linkerd-multicluster` and extension namespaces • Limit the ability to read secrets or modify critical resources, such as the `linkerd-identity-trust-roots` `ConfigMap`, to designated cluster administrators and GitOps service accounts to prevent control plane tampering	AC-3: Access Enforcement AC-6: Least Privilege CM-5: Access Restrictions for Change
R-07: Restrict and Audit Access to Secrets	High	T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys T-13 - Remote cluster Kubernetes API access via link credentials theft	• Limit RBAC access to Kubernetes `Secret` resources, particularly those containing private keys, CA material, or multicluster link credentials	AC-3: Access Enforcement AC-6: Least Privilege

R-08: Least Privilege Access to Linkerd Policy Resources	High	T-05 - Unauthorised modification of Linkerd policy resources	• Limit RBAC permissions for creating, updating, or deleting Linkerd policy Custom Resources (`Server`, `AuthorizationPolicy`, `HTTPRoute` and others)	AC-3: Access Enforcement AC-6: Least Privilege CM-5: Access Restrictions for Change
R-09: Encrypt and Protect Kubernetes Secrets at Rest	High	T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys T-13 - Remote cluster Kubernetes API access via link credentials theft	• Configure the Kubernetes API server to encrypt secrets at rest using an external KMS or envelope encryption provider • This protects sensitive infrastructure data, such as the active Linkerd issuer certificate and associated private keys, from being extracted in the event of `etcd` or host-level node compromise	SC-28: Protection of Information at Rest SC-13: Cryptographic Protection
R-10: Enforce Default-Deny Network Policies	High	T-26 - Overly permissive or absent EgressNetwork policies T-12 - Cross-cluster lateral movement in flat network multicluster mode	• Apply Kubernetes `NetworkPolicy` resources to establish a baseline default-deny posture at the CNI network level	AC-4: Information Flow Enforcement SC-7: Boundary Protection
R-11: Workload Runtime Security Hardening	High	T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation	• Explicitly define a restrictive `securityContext` for all pods and individual containers	CM-2: Baseline Configuration

		T-23 - Workload identity spoofing via proxy credential theft T-27 - Plaintext data interception via local loopback snooping	• Set `capabilities.drop: ["ALL"]` to strip the container of unnecessary Linux kernel privileges • Configure the `RuntimeDefault seccomp` profile for workloads to block dangerous or unnecessary kernel system calls • Enforce `runAsNonRoot: true` and `allowPrivilegeEscalation: false` to prevent attackers from executing code as root or escalating process privileges • Set `readOnlyRootFilesystem: true` to block attackers from writing malicious payloads or modifying system files on disk	CM-6: Configuration Settings CM-7: Least Functionality SC-39: Process Isolation
R-12: Centralise Policy Management via GitOps	High	T-15 - Cross-cluster policy drift	• Manage Linkerd `Server`, `AuthorizationPolicy`, and other policy resources through a centralised GitOps workflow (e.g., Flux or Argo CD) • Sourcing configurations from a single repository ensures access controls remain synchronised across linked clusters and automatically remediates configuration drift	CM-2: Baseline Configuration CM-3: Configuration Change Control SA-10: Developer Configuration Management

R-13: Certificate Lifecycle Management and Automation	High	T-17 - Mesh-wide outage via trust anchor or identity issuer certificate expiration T-19 - Mesh management outage via webhook TLS certificate expiration	• Implement robust lifecycle management and automation for all cryptographic materials • Use tools like `cert-manager` to handle the issuance and continuous rotation of the intermediate identity issuer and Linkerd admission webhook certificates • For the mesh trust anchor, establish an automated procedure or runbook for its rotation. Trust anchor rotation is a multi-step process requiring the distribution of the new bundle alongside the existing one, followed by cluster-wide workload restarts so proxy sidecars adopt the new cryptographic context	SC-12: Cryptographic Key Establishment and Management SC-17: Public Key Infrastructure Certificates
R-14: Implement Multi-Cluster Network Policies in Flat Network Architectures	High	T-12 - Cross-cluster lateral movement in flat network multicluster mode	• In flat network multicluster environments, deploy a CNI with multi-cluster network policy capabilities (such as Cilium with Cluster Mesh) to enforce network-level isolation	AC-4: Information Flow Enforcement SC-7: Boundary Protection SC-46: Cross Domain Policy Enforcement
R-15: Least Privileged Multicluster Link Credentials	High	T-13 - Remote cluster Kubernetes API access via link credentials theft	• Apply least privilege RBAC to the `ServiceAccount` used by the remote multicluster service mirror controller	AC-3: Access Enforcement AC-6: Least Privilege

		T-14 - Data exposure and compliance violations via inappropriate service mirroring or federation	• Instead of granting cluster-wide read access for service discovery, bind the credentials only to the specific namespaces intended for service mirroring or federation to limit the blast radius of remote cluster compromise	
R-16: Restrict Service Mirroring and Federation via Admission Controller	High	T-14 - Data exposure and compliance violations via inappropriate service mirroring or federation	• Use validating admission controllers to restrict the application of the `mirror.linkerd.io/exported` and `mirror.linkerd.io/federated` labels • Limit these labels to approved namespaces to prevent sensitive internal workloads from being erroneously or maliciously exposed across disparate trust boundaries in a multicluster setup	SI-10: Information Input Validation SC-46: Cross Domain Policy Enforcement
R-17: Restrict Multicluster Gateway Access Controls and Network Exposure	High	T-11 - Cross-cluster identity obfuscation via gateway confused deputy T-21 - Multicluster gateway overexposure and denial of service	• In hierarchical multicluster deployments, define `AuthorizationPolicy` and default-deny Kubernetes Network Policies that restrict inbound gateway access • Explicitly restrict which remote cluster identities (via `MeshTLSAuthentication`) and source networks (via	SC-5: Denial-of-service Protection SC-7: Boundary Protection

			`NetworkAuthentication`) are permitted to communicate with the gateway • Note the the default `linkerd-multicluster` Helm chart is designed as a quick start and, due to the permissive policies defined for the gateway in it, should not be used as a production blueprint • Configure cloud provider firewall rules to restrict ingress traffic targeting the Linkerd multicluster gateway `LoadBalancer`; permit traffic only from the known CIDR blocks or NAT gateway / egress IP addresses of trusted remote clusters	
R-18: Configure Ingress Controller to Strip `l5d-dst-override` Header	High	T-07 - Ingress mode proxy open relay via spoofed l5d-dst-override header	• Configure edge ingress controllers to strip the `l5d-dst-override` HTTP header from all incoming external requests • This prevents external attackers from using the header to spoof routing destinations and bypass ingress path rules to reach arbitrary internal, unexposed services or external endpoints.	AC-4: Information Flow Enforcement SC-7: Boundary Protection SI-10: Information Input Validation

R-19: Remove Excessive Prometheus Privileges	High	T-16 - Cluster compromise via Linkerd-Viz bundled Prometheus nodes/proxy GET permissions	• Remove the overly permissive `nodes/proxy GET` permission from the `ClusterRole` associated with the bundled Linkerd-viz Prometheus instance • Instead, leverage the fine-grained kubelet API authorisation (generally available in Kubernetes v1.36) to grant Prometheus access only to the required `kubelet` metrics endpoints (such as `/metrics` and `/metrics/cadvisor`)	AC-6: Least Privilege CM-7: Least Functionality
R-20: Deploy Linkerd Control Plane in High Availability Mode	High	T-18 - Linkerd control plane denial of service	• Install Linkerd in High Availability (HA) mode for production environments • This provisions multiple replicas of core control plane components and applies pod anti-affinity rules and Pod Disruption Budgets to distribute them across distinct nodes, ensuring mesh resilience during node failures or updates	CM-6: Configuration Settings SC-5: Denial-of-service Protection SC-6: Resource Availability
R-21: Enforce Pod Security Standards (PSS)	Medium	T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation T-23 - Workload identity spoofing via proxy	• Where operationally feasible, apply a PSS Baseline or Restricted profile (or equivalent admission policies) for namespaces	CM-2: Baseline Configuration CM-7: Least Functionality SC-39: Process Isolation

		credential theft T-27 - Plaintext data interception via local loopback snooping		
R-22: Enable the Linkerd CNI Plugin	Medium	T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation T-27 - Plaintext data interception via local loopback snooping	• Deploy the Linkerd CNI plugin to manage the configuration of proxy `iptables` routing rules at the node level • This removes the requirement to run the `linkerd-init` container with `NET_ADMIN` capabilities and root privileges within every meshed pod, allowing for the implementation of other mitigations such as R-21: Enforce Pod Security Standards (PSS) and R-11: Workload Runtime Security Hardening in order to reduce the risk of deploying workloads with insecure configurations	CM-6: Configuration Settings CM-7: Least Functionality SC-3: Security Function Isolation SC-39: Process Isolation
R-23: Opt Out of Service Account Token Automounting	Medium	T-06 - Identity spoofing via Kubernetes Service Account token theft or RBAC abuse	• Set `automountServiceAccountToken: false` in `Pod` and `ServiceAccount` resources unless required	CM-2: Baseline Configuration CM-6: Configuration Settings AC-6: Least Privilege CM-7: Least Functionality

R-24: Enforce `ExternalNetwork` Policies for Egress Traffic	Medium	T-26 - Overly permissive or absent EgressNetwork policies	• Use Linkerd `ExternalNetwork` Custom Resources to define and restrict permitted outbound traffic from the mesh	AC-4: Information Flow Enforcement SC-7: Boundary Protection
R-25: Restrict Linkerd-Viz Tap API Access	Medium	T-28 - Linkerd-Viz Tap API abuse for real-time traffic interception	• Revoke broad access to the Linkerd-Viz tap API by restricting the Linkerd dashboard's tap access when installing the extension via `dashboard.restrictPrivileges` and limiting the `linkerd-viz-tap-admin ClusterRole` to authorised diagnostic accounts	AC-3: Access Enforcement AC-6: Least Privilege AU-9: Protection of Audit Information
R-26: Exclude Sensitive Headers from Being Captured by Linkerd-Viz Tap	Medium	T-28 - Linkerd-Viz Tap API abuse for real-time traffic interception	• Configure the `tap.ignoreHeaders` Helm value for Linkerd-Viz to redact sensitive HTTP headers, such as `Authorization`, from diagnostic outputs	AU-3: Content of Audit Records
R-27: Explicitly Declare Application Protocols for Linkerd Server Resources and Kubernetes Services	Medium	T-24 - L7 policy evasion via protocol detection bypass or opaque port misconfiguration	• Set the `proxyProtocol` field in Linkerd `Server` resources or use the `appProtocol` field on standard Kubernetes `Service` manifests • Explicitly declaring the protocol disables dynamic protocol sniffing, preventing attackers from exploiting detection timeouts or inducing	CM-7: Least Functionality

			protocol confusion to bypass L7 policy enforcement	
R-28: Configure Robust Logging and Monitoring	Medium	T-01 - Identity forgery and workload impersonation via extraction of in-cluster trust anchor or identity issuer keys T-09 - Linkerd control plane compromise T-10 - Identity spoofing or denial of service via trust bundle tampering T-11 - Cross-cluster identity obfuscation via gateway confused deputy T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation	• Enable Kubernetes API audit logging and forward logs to a centralised SIEM • Configure real-time alerts for any access to, or modification of, Kubernetes Secrets within the `linkerd` and associated mesh namespaces to protect critical cryptographic materials like trust anchors and issuer certificates • Set an appropriate baseline log level (such as `info`) for the Linkerd control plane components and data plane proxies to ensure security-relevant events are captured without generating excessive noise • Enable audit logging on the multi-cluster gateway proxy in hierarchical deployments to log the original cryptographic identity of inbound cross-cluster requests • Monitor Linkerd telemetry for anomalous traffic patterns	AU-2: Event Logging SI-4: System Monitoring

R-29: Implement Application Level Authorisation for Sensitive Services	Medium	T-06 - Identity spoofing via Kubernetes Service Account token theft or RBAC abuse T-23 - Workload identity spoofing via proxy credential theft T-24 - L7 policy evasion via protocol detection bypass or opaque port misconfiguration T-25 - L7 policy evasion via proxy-application parser differentials	• Implement application-layer authentication and authorisation for sensitive services as a defence-in-depth measure in the event of a legitimate mesh-level mTLS identity being compromised or impersonated, or L7 policies bypassed	AC-3: Access Enforcement
R-30: Deploy a Runtime Security Solution	Medium	T-08 - Data plane evasion via pod-level misconfigurations or privilege escalation T-23 - Workload identity spoofing via proxy credential theft	• Implement a kernel-level runtime security tool, such as Falco or Tetragon, to monitor container behaviour and detect post-exploitation activity • Use this to alert on anomalous events that indicate attempts to bypass the mesh or extract credentials, such as unauthorised modifications to `iptables`, the creation of raw network sockets, `ptrace` system calls, or unexpected access to	SI-3: Malicious Code Protection SI-4: System Monitoring

			sensitive files or in-memory `tmpfs` mounts	
R-31: Configure Short-Lived Workload Certificates	Low	T-23 - Workload identity spoofing via proxy credential theft	• Ensure that a short TTL is set for workload certificates in the Linkerd control plane configuration, **not exceeding the secure default 24 hours**	SC-12: Cryptographic Key Establishment and Management
R-32: Ensure the Proxy Shutdown Endpoint is Disabled	Low	T-30 - Pod denial of service via proxy shutdown endpoint	• Ensure the Linkerd proxy's shutdown endpoint is disabled **(disabled by default since Linkerd 2.16)**	CM-6: Configuration Settings CM-7: Least Functionality SC-5: Denial-of-service Protection
R-33: Ensure Hostname Labels for Metrics are Disabled	Low	T-29 - Denial of service and infrastructure overwhelm via telemetry resource exhaustion (CVE-2025-43915)	• Ensure the inclusion of hostname labels within Linkerd proxy metrics is disabled **(disabled by default via `proxy.metrics.hostnameLabels: false`)** to avoid potential telemetry resource exhaustion due to high cardinality metrics	CM-6: Configuration Settings CM-7: Least Functionality SC-5: Denial-of-service Protection

Appendix

Considerations for Buoyant Enterprise for Linkerd

Buoyant Enterprise for Linkerd (BEL) is the commercial, fully supported distribution of the Linkerd service mesh, provided by the project's original creators. BEL introduces several operational, security, and compliance features aimed at enterprise environments.

From a threat modelling perspective, introducing BEL alters the risk landscape by automating complex, error-prone administrative tasks and enforcing stronger supply chain security. Below is an assessment of how BEL capabilities intersect with and mitigate the threats identified in this document, alongside broader enhancements to Linkerd's operational resilience, security posture, and regulatory compliance.

Automated trust anchor rotation

As identified in the threat model, the expiration or manual rotation of the trust anchor certificate is a critical event (T-17, T-20). Manual rotation requires a fragile, multi-step process where skipping a step or executing one out of order can break mutual mTLS validation across the cluster, resulting in a self-inflicted denial of service.

BEL addresses this via a dedicated `trust-rotation-operator`. Rather than relying on manual administrative runbooks, the operator safely automates the sequence: deploying the updated trust bundle, orchestrating the rollout across control plane and data plane components, and retiring the old anchor once convergence is verified.

Authorisation policy automation

A frequent vulnerability pattern in service mesh deployments is the reliance on permissive, default-allow policy configurations (T-02). This usually occurs because defining and maintaining explicit, fine-grained authorisation policies for complex microservice architectures is operationally burdensome.

BEL includes a policy automation feature designed to bridge this gap. By profiling observed traffic within a functioning application environment, the tool automatically generates corresponding `Server`, `MeshTLSAuthentication`, and `AuthorizationPolicy` resources. This significantly lowers the operational friction of implementing strict zero-trust policies, as well as keeping them up to date.

Vulnerability management, lifecycle automation, and supply chain security

The compromise of any Linkerd control plane component (T-09) or the data plane proxy (T-23) can have a serious impact on the integrity of the mesh and zero-trust policy enforcement.

Therefore, the security posture of the mesh relies on the rapid, safe application of security patches and the verifiable integrity of its release artifacts. BEL addresses these operational challenges in three key ways:

- **CVE and Security Patch Policy:** Buoyant guarantees that vulnerabilities classified as Medium severity or higher are actively patched in supported, stable minor releases. This allows organisations to remediate known vulnerabilities rapidly without being forced to adopt bleeding-edge, potentially unstable edge releases
- **Lifecycle Automation:** BEL manages installations, upgrades, and rollbacks via a GitOps-compatible Kubernetes operator. By defining the desired version declaratively, the operator orchestrates the safe, uniform rollout of security patches across both the control plane and all data plane proxies, eliminating human error during complex upgrade cycles
- **Supply Chain Attestations:** To defend against supply chain attacks, every BEL release ships with signed artefacts, Software Bills of Materials (SBOM), and SLSA Build Provenance attestations. This enables security teams to programmatically verify the integrity and origin of the Linkerd Helm charts and images before they are deployed to the cluster

FIPS-verified builds

For deployments operating within regulated industries (such as the US federal government, defense, or healthcare sectors), compliance with specific cryptographic standards is often mandatory. BEL provides builds of Linkerd that utilise FIPS-validated cryptographic modules. This ensures that all encryption meets FIPS-140-2 and FIPS-140-3 standards without requiring any code alterations at the application layer.

References

Linkerd
https://linkerd.io/

Linkerd Documentation
https://linkerd.io/docs/getting-started/index.html

2022 Linkerd Threat Model by Trail of Bits (single cluster only)
https://github.com/trailofbits/publications/blob/master/reviews/Linkerd-threatmodel.pdf

NIST SP 800-233 Service Mesh Proxy Models for Cloud-Native Applications
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-233.pdf

About

ControlPlane is a global cloud native and open source cybersecurity consultancy operating in London, New York, and Auckland. We have industry-leading expertise in the architecture, audit, and implementation of zero trust infrastructure for regulated industries. With a deep understanding of secure-by-design and secure-by-default cloud, Kubernetes, and supply chain security we conduct threat modeling, penetration testing, and cloud native security training to the highest standard.

ControlPlane is trusted as the partner of choice in securing: multinational banks; major public clouds; international financial institutions; critical national infrastructure programs; multinational oil and gas companies, healthcare and insurance providers; and global media firms.

https://control-plane.io/

Team

Tom Cope
Gabriela Georgieva
Samuel Holmes

Reviewers

Ehsan Shirazi
William Morgan
Eric Douglas