Check for updates

NIST Special Publication 800
NIST SP 800-233

Service Mesh Proxy Models for Cloud-Native Applications

Ramaswamy Chandramouli
Zack Butcher
James Callaghan

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-233

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Special Publication 800
NIST SP 800-233

Service Mesh Proxy Models for Cloud-Native Applications

Ramaswamy Chandramouli
Computer Security Division
Information Technology Laboratory

Zack Butcher
Tetrate, Inc.

James Callaghan
control-plane.io, Inc.

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-233

October 2024



U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Certain commercial equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

Authority
This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3551 et seq., Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History
Approved by the NIST Editorial Review Board on 2024-10-11

How to Cite this NIST Technical Series Publication:
Chandramouli R, Butcher Z, Callaghan J (2024) Service Mesh Proxy Models for Cloud-Native Applications. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-233. https://doi.org/10.6028/NIST.SP.800-233

Author ORCID iDs
Ramaswamy Chandramouli: 0000-0002-7387-5858

Contact Information

sp800-233-comments@nist.gov

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

Additional Information

Additional information about this publication is available at https://csrc.nist.gov/pubs/sp/800/233/final, including related content, potential updates, and document history.

All comments are subject to release under the Freedom of Information Act (FOIA).

Abstract

The service mesh has become the de facto application services infrastructure for cloud-native applications. It enables the various runtime functions of an application through proxies that form the data plane of the service mesh. Depending on the distribution of the network layer functions and the granularity of association of the proxies to individual services and computing nodes, different proxy models or data plane architectures have emerged. This document describes a threat profile for each of the data plane architectures with a detailed threat analysis to make recommendations on their applicability for cloud-native applications with different security risk profiles.

Keywords

cloud-native application; data plane architecture; proxy model; service mesh; threat profile.

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Patent Disclosure Notice

NOTICE: ITL has requested that holders of patent claims whose use may be required for compliance with the guidance or requirements of this publication disclose such patent claims to ITL. However, holders of patents are not obligated to respond to ITL calls for patents and ITL has not undertaken a patent search in order to identify which, if any, patents may apply to this publication.

As of the date of publication and following call(s) for the identification of patent claims whose use may be required for compliance with the guidance or requirements of this publication, no such patent claims have been identified to ITL.

No representation is made or implied by ITL that licenses are not required to avoid patent infringement in the use of this publication.

Table of Contents

Executive Summary **1**

1. Introduction **2**

1.1. L4 and L7 Functions of Proxies 3

1.2. Objective and Target Audience 3

1.3. Relationship to Other NIST Documents 4

1.4. Document Structure 4

2. Typical Service Mesh Data Plane Capabilities and Associated Proxy Functions **5**

3. Proxy Models (Data Plane Architectures) in Service Mesh Implementations **7**

3.1. L4 and L7 Proxy per Service Instance (DPA-1) — Sidecar Model 7

3.2. Shared L4 – L7 per Service Model (DPA-2) 8

3.3. Shared L4 and L7 Model (DPA-3) 9

3.4. L4 and L7 as Part of the Application Model (DPA-4) 10

4. Data Plane Architecture Threat Scenarios and Analysis Methodology **12**

4.1. Threat Analysis Methodology 13

5. Detailed Threat Analysis for Data Plane Architectures **14**

5.1. Threat Analysis for L4 and L7 Proxy per Service Instance (DPA-1) — Sidecar Model 14

5.1.1. Compromised L4 Proxy (TR-1) 14

5.1.2. Compromised Application Container (TR-2) 14

5.1.3. Compromise of Business Data (TR-3) 15

5.1.4. Compromised L7 Proxy (TR-4) 15

5.1.5. Compromise of Shared L7 Proxy (TR-5) 15

5.1.6. Outdated Client Libraries in Applications (TR-6) 16

5.1.7. Denial of Service (TR-7) 16

5.1.8. Resource Consumption (TR-8) 17

5.1.9. Privileged L4 Proxy (TR-9) 17

5.1.10. Data Plane (Service Mesh) Bypassed (TR-10) 17

5.1.11 Overall Threat Score 18

5.2. Threat Analysis for Shared L4 – L7 per Service Model (DPA-2) 18

5.2.1. Compromised L4 Proxy (TR-1) 18

5.2.2. Compromised Application Container (TR-2) 18

5.2.3. Compromise of Business Data (TR-3) 18

5.2.4. Compromised L7 Proxy (TR-4) 19

5.2.5. Compromise of Shared L7 Proxy (TR-5) 19

5.2.6. Outdated Client Libraries in Applications (TR-6) 19

5.2.7. Denial of Service (TR-7) 19
5.2.8. Resource Consumption (TR-8) 20
5.2.9 Privileged L4 Proxy (TR-9) 20
5.2.10 Data Plane (Service Mesh) Bypassed (TR-10) 21
5.2.11 Overall Threat Score 21
5.3. Threat Analysis for Shared L4 and L7 Model (DPA-3) 21
5.3.1. Compromised L4 Proxy (TR-1) 21
5.3.2. Compromised Application Container (TR-2) 22
5.3.3. Compromise of Business Data (TR-3) 22
5.3.4. Compromised L7 Proxy (TR-4) 22
5.3.5. Compromise of Shared L7 Proxy (TR-5) 23
5.3.6. Outdated Client Libraries in Applications (TR-6) 23
5.3.7. Denial of Service (TR-7) 23
5.3.8. Resource Consumption (TR-8) 23
5.3.9. Privileged L4 Proxy (TR-9) 24
5.3.10. Data Plane (Service Mesh) Bypassed (TR-10) 24
5.3.11 Overall Threat Score 24
5.4. Threat Analysis for L4 and L7 as Part of the Application Model (gRPC proxyless Model (DPA-4)) 25
5.4.1. Compromised L4 Proxy (TR-1) 25
5.4.2. Compromised Application Container (TR-2) 25
5.4.3. Compromise of Business Data (TR-3) 25
5.4.4. Compromised L7 Proxy (TR-4) 25
5.4.5. Compromise of Shared L7 Proxy (TR-5) 26
5.4.6. Outdated Client Libraries in Applications (TR-6) 26
5.4.7. Denial of Service (TR-7) 26
5.4.8. Resource Consumption (TR-8) 27
5.4.9 Privileged L4 Proxy (TR-9) 27
5.4.10 Data Plane (Service Mesh) Bypassed (TR-10) 27
5.4.11 Overall Threat Score 28
6. Recommendations Based on the Application Security Risk Profile 29
6.1 Cloud-Native Applications With Low Risk Profiles 29
6.2 Cloud-Native Applications with Medium Risk Profiles 30
6.3 Cloud-Native Applications With High Risk Profiles 30
7. Summary and Conclusions 32

References.....33

Acknowledgments

The authors would like to express their thanks to Francesco Beltramini of control-plane.io for participating in discussions and providing his valuable perspective. The authors would also like to express their thanks to Isabel Van Wyk of NIST for her detailed editorial review, both for the public comment version as well as for the final publication

Executive Summary

A centralized infrastructure called a service mesh can provide run-time services for cloud-native applications that consist of multiple loosely coupled components called microservices. These services include secure communication, service discovery, resiliency, and authorization of application communication. These services are mainly provided through *proxies* that form the *data plane* of the service mesh, which is the layer that handles application traffic at runtime and enforces policy.

The functions that the proxies provide can be broadly categorized into two groups based on the Open Systems Interconnection (OSI) model's network layer to which those functions pertain: Layer 4 ("L4") and Layer 7 ("L7"). In most service mesh deployments in production environments today, all proxy functions that provide services in both L4 and L7 layers are packed into a single proxy that is assigned to a single microservice. This service mesh proxy model is called a *sidecar proxy model* since the proxy is not only associated with a single service but is implemented to execute in the same network space as the service.

However, performance and resource considerations have led to the exploration of alternate proxy models that involve splitting L4 and L7 functions into different proxies and the association or assignments of these proxies to either a single service or a group of services. This enables the proxies to be implemented at different locations at the granularity of a node rather than at the level of services. Though different models are theoretically possible, this document only considers service mesh proxy models in the data plane implementation of commonly used service mesh offerings at different stages.

Various potential or likely threats to proxy functions may result in different types of exploits in different proxy models. This variation is due to several factors, such as the attack surface (i.e., communication patterns to which a particular proxy is exposed), the number of clients (services) served, and the OSI layer functions that they provide (e.g., L7 functions are more complicated and likely to have more vulnerabilities than L4 functions). The two main contributions of this document are the following:

1. The nature of the exploits that are possible for each threat in each of the proxy models is characterized by assigning scores to the impact and likelihood of each of the threats in each of the proxy models or architectural patterns, resulting in a threat profile that is associated with each architectural pattern or proxy model of service mesh.
2. Each threat profile has an inherent set of security trade-offs at an architectural level. The implications of these trade-offs in meeting the requirements associated with the security risk profiles of different cloud-native applications are analyzed to make a broad set of recommendations toward specific architectural patterns that are appropriate for applications with different security risk profiles.

1. Introduction

"Cloud-native" refers to an architectural philosophy for building scalable, resilient systems that are designed to leverage the advantages of cloud computing environments. Cloud-native applications can run both on-premises and in public cloud platforms and are normally built using agile development methodologies, such as continuous integration/continuous delivery (CI/CD). Typically, technologies such as containerization and virtual machines (VMs) are used, and resilience and fail-safe features will be built in.

Microservices-based applications use an architectural approach in which the entire application is broken into loosely coupled components that can be independently updated and scaled. The implementation of microservices is enabled using containers that in turn require orchestration tools and often employ a centralized services infrastructure (e.g., service mesh) to provide all runtime application services, including network connectivity, security, resiliency, and monitoring capabilities. Microservices-based applications can be implemented and deployed as cloud-native, though they represent an independent architectural approach.

The infrastructure services or functions provided by a service mesh during application runtime are provided by entities called proxies, which constitute the data plane of the service mesh. In addition, the service mesh consists of another architectural component called the control plane, which supports the functions of the data plane through interfaces to define configurations, inject software programs, and provide security artifacts (e.g., certificates).

Various configurations for proxies are being developed and tested based on the performance and security assurance data obtained during the deployment of service mesh over the last several years. These configurations are proxy (implementation) models that are based on the OSI layer functions that they provide (described in the following paragraphs) and the granularity of association between a proxy and services. Since proxies are the predominant entities of the data plane of a service mesh, these various proxy models are also called data plane architectures.

The OSI model [1] is a useful abstraction for thinking about the functions required to serve an application over the network. It describes seven "layers," from the physical wires that connect two machines (i.e., Layer 1 – L1, the physical layer) to the application itself (i.e., Layer 7 – L7, the application layer).

Layers 3, 4, and 7 are key to facilitating communication between cloud-native applications (e.g., two microservices making Hypertext Transfer Protocol (HTTP)/REST calls to each other):

- Layer 3 ("L3"), the network layer, facilitates baseline connectivity between two workloads or service instances. In nearly all cases, the Internet Protocol (IP) is used as the L3 implementation.
- Layer 4 ("L4"), the transport layer, facilitates the reliable transmission of data between workloads on the network. It also includes capabilities like encryption. Transport Control Protocol (TCP) and User Datagram Protocol (UDP) are commonly used L4 implementations, where transport layer security (TLS) provides encryption.

- Layer 7 ("L7"), the application layer, is where protocols like HTTP operate — in user applications themselves (e.g., HTTP web servers, Secure Shell (SSH) servers).

With respect to the layers above, a service mesh's proxies in cloud-native environments are:

- Agnostic to L3 if the microservice instances can communicate at L3 and the proxy can communicate with the mesh's control plane.
- At Layer 4 (L4): Connection establishment, management, and resiliency (e.g., connection-level retries); TLS (encryption in transit); application identity, authentication, and authorization; access policy based on network 5-tuple (e.g., source IP address and port, destination IP address and port, and transport protocol).
- At Layer 7 (L7): Service discovery, request-level resiliency (e.g., retries, circuit breakers, outlier detection); and application observability.

1.1. L4 and L7 Functions of Proxies

There are two key aspects of proxy models:

1. Proxy functions: The functions that a service mesh's proxies provide can be broadly categorized into two groups based on the OSI model's layer [1] to which those functions pertain: Layer 4 ("L4") and Layer 7 ("L7"). The associated proxies are called L4 proxies and L7 proxies, respectively. The study of proxy functions requires an understanding of the OSI's L4 and L7 layers from the network stack point of view and the specific network services provided by those layers.
2. Granularity of association: A proxy can be associated with a single microservice instance, an entire service, or deployed to provide functions for a group of services. Depending on the nature of this association, a proxy may execute within the same network space as the service, at the same node where the group of services to which it caters run, or in an independent node dedicated to proxies where no application services run.

1.2. Objective and Target Audience

This document will give a brief overview of the four data plane architectures (proxy models) being pursued by a range of service mesh implementations today. It will also provide threat profiles for different proxy models with a detailed threat analysis that involves 10 types of common threats. These threat profiles will inform recommendations regarding their applicability (usage) for cloud-native applications with different security risk profiles.

The target audience for these recommendations includes:

- Infrastructure owners, platform/infrastructure engineers, and their team leaders who build and deploy secure runtime environments for applications by choosing the right architecture for their environment given the risk factors of the applications that they will be running and the resulting security risk profile.

- Personnel in charge of infrastructure operations who need to be familiar with the various building blocks of the proxy models or data plane architectures (and their associated functions and interactions) to troubleshoot in the event of performance (i.e., availability) and security issues

1.3. Relationship to Other NIST Documents

This document can be used as an adjunct to the NIST Special Publication (SP) 800-204 series of publications [2][3][4][5], which offer guidance on providing security assurance for cloud-native applications integrated with a service mesh from the following perspectives: strategy, configuration, and development/deployment paradigm. However, this document focuses on the various configurations of the application service infrastructure elements (i.e., proxies) and the resulting architectures (i.e., data plane architecture of the service mesh) that have different security implications for the application that is hosted under each of these configurations.

1.4. Document Structure

This document is organized as follows:

- Section 2 lists the typical capabilities of the data plane of the service mesh under three headings (i.e., security, observability, and traffic management) and the corresponding L4 and L7 proxy functions implemented under those capabilities.
- Section 3 provides a brief overview of the four proxy models or data plane architectures.
- Section 4 discusses proxy model threat scenarios and the threat analysis methodology adopted in this document for evaluating the threat profile score for the four data plane architectures.
- Section 5 provides a detailed threat analysis for the four data plane architectures by assigning scores to the impact and likelihood factors associated with each threat and using them to arrive at an overall threat score.
- Section 0 provides recommendations on the applicability (usage) of each of the four data plane architectures for cloud-native applications of different security risk profiles based on their security requirements.
- Section 0 provides the summary and conclusions.

2. Typical Service Mesh Data Plane Capabilities and Associated Proxy Functions

This document's methodology examines the security trade-offs of the proxy models (i.e., data plane architectures) and the implementations of the various capabilities that result as L4 and L7 functions in proxies. Determining the totality of proxy functions requires an analysis of each capability, the category it falls under, and the granularity of the function that it provides at L4 and L7 levels.

Table 1 - Security Capabilities [15]

Capability	L4 Function(s)	L7 Function(s)
Service-to-service authentication	SPIFFE, via mTLS certs. Control plane issues a short-lived X.509 encoding the pod's service account identity.	N/A—service identity in a service mesh is usually based on TLS only.
Service-to-service authorization	Network-based authorization, plus identity-based policy, e.g.: *A* can accept inbound calls from only "10.2.0.0/16"; *A* can call *B.*	Full policy, e.g.: *A* can GET /foo on *B* only with valid end-user credentials containing the READ scope.
End-user authentication	N/A—we can't apply per-user settings.	Local authentication of JWTs, support for remote authentication via OAuth and OIDC flows.
End-user authorization	N/A—see above.	Service-to-service policies can be extended to require end user credentials with specific scopes, issuers, principal, audiences, etc. but it cannot be used for full user-to-resource access control. Full user-to-resource access should be implemented using external authorization.
Mesh proxy's External Authorization Application Programming Interface (API) (ext_authz)	Cannot perform any per-request policy; ext_authz API is only configurable for L7 traffic.	Enforce per-request policy with decisions from an external service, e.g., Open Policy Agent (OPA).

Table 2 - Observability Capabilities [15]

Capability	L4 Function(s)	L7 Function(s)
Logging	Basic network information: network 5-tuple, bytes sent/received, etc.	Full request metadata logging, in addition to basic network information.
Tracing	Not today; potentially possible in the future with emerging technology like HBONE.	Mesh data plane participates in distributed tracing.
Metrics	TCP only (bytes sent/received, number of packets, etc.).	L7 Rate Errors Duration (RED) metrics: rate of requests, rate of errors, request duration (latency).

Table 3 – Traffic Management Capabilities [15]

Capability	L4 Function(s)	L7 Function(s)
Load balancing	Connection level only.	Per request, enabling, e.g., canary deployments, gRemote Procedure Call (gRPC) traffic.
Circuit breaking	TCP only.	HTTP settings in addition to TCP.
Outlier detection	On connection establishment/failure.	On request success/failure.
Rate limiting	Rate limit on L4 connection data only, on connection establishment, with global and local rate limiting options.	Rate limit on L7 request metadata, per request.
Timeouts	Connection establishment only (connection keep-alive is configured via circuit breaking settings).	Timeouts per request, and per retry.
Retries	Retry connection establishment	Retry per request failure.
Fault Injection	N/A—fault injection cannot be configured on TCP connections.	Full application- and connection-level faults (timeouts, delays, specific response codes).
Traffic Mirroring	N/A—HTTP only	Percentage-based mirroring of requests to multiple backends.

L7 functions that are carried out by proxies are much more complex than L4 functions as the latter are carried out in lower layers of the OSI stack and involve protocols such as IP and TCP. For example, parsing a TCP stream for L4 functionality simply requires decoding a fixed set of bytes as integers (i.e., the packet header), while handling HTTP requests for L7 functionality requires decoding HTTP headers, including complex string parsing and compression with variable amounts of data. Additionally, the data dealt with in an L7 function are user-supplied (i.e., can be controlled by an attacker), while the TCP data at L4 are typically system-supplied as part of routing a request to the infrastructure. This means that there is less room to embed malicious data without breaking the system itself. In one case study, the proxy Envoy is used as the data plane by several service mesh implementations. Historically, majority of Envoy vulnerabilities have been in L7 function-related code compared to L4 function-related code.

3. Proxy Models (Data Plane Architectures) in Service Mesh Implementations

Different data plane architectures or proxy models in the service mesh are consequences of the following parameters:

- Delineation of L4 and L7 functions
- Nature of association of a proxy to service instances (1:1 or 1:N)

This section provides an overview of the building blocks of different data plane architectures to facilitate the threat analysis in Sec. 5. Iterations of the different architectures were driven by the adoption of mesh across a variety of use cases, necessitating trade-offs in terms of performance, reliability, and security for organizations with different application risk profiles. In spite of these different operating scenarios, the sidecar model has been the primary method for delivering service mesh capabilities.

The various alternate data plane architectures, including the one with widespread deployment at present, are:

- L4 and L7 per Service Instance (DPA-1) — Sidecar model
- Shared L4 – L7 per Service (DPA-2) — A shared L4 proxy per node (i.e., shared among all applications that execute on the same physical host) with L7 proxies dedicated per service account or namespace.
- Shared L4 and L7 Model (DPA-3) — A shared L4 *and* L7 proxy per node (i.e., shared among all applications on the same physical host)
- L4 and L7 as Part of the Application (gRPC proxy-less model [14]) (DPA-4) — Both L4 and L7 functions are part of the application server itself (e.g., gRPC, Java Spring [16]) rather than implemented in stand-alone proxies.

Though the last architectural pattern does not have distinct entities (e.g., proxies), all of the service mesh capabilities delivered by proxies are enabled by these frameworks.

3.1. L4 and L7 Proxy per Service Instance (DPA-1) — Sidecar Model

The most common service mesh data plane architecture is called the “sidecar model” because the proxy sits beside every instance of every service. The model dedicates a proxy that has the capability to implement both L4 and L7 functions for each application (service) instance. In its security model, the proxy holds one identity for the service it is deployed beside and resides in the same trust domain as the application (e.g., in Kubernetes, it exists in the same pod; on a virtual machine (VM), it is deployed in the same VM as the service itself). The service and the proxy communicate with each other through the “local host interface” rather than through a network socket. However, the proxy presents a larger attack surface than the service because it implements complex L7 functions. An example of a data plane architecture is one that is implemented in the Istio service mesh with an envoy proxy deployed per pod that performs both L4 and L7 functions.

A schematic diagram of this architecture is shown in Fig. 1.

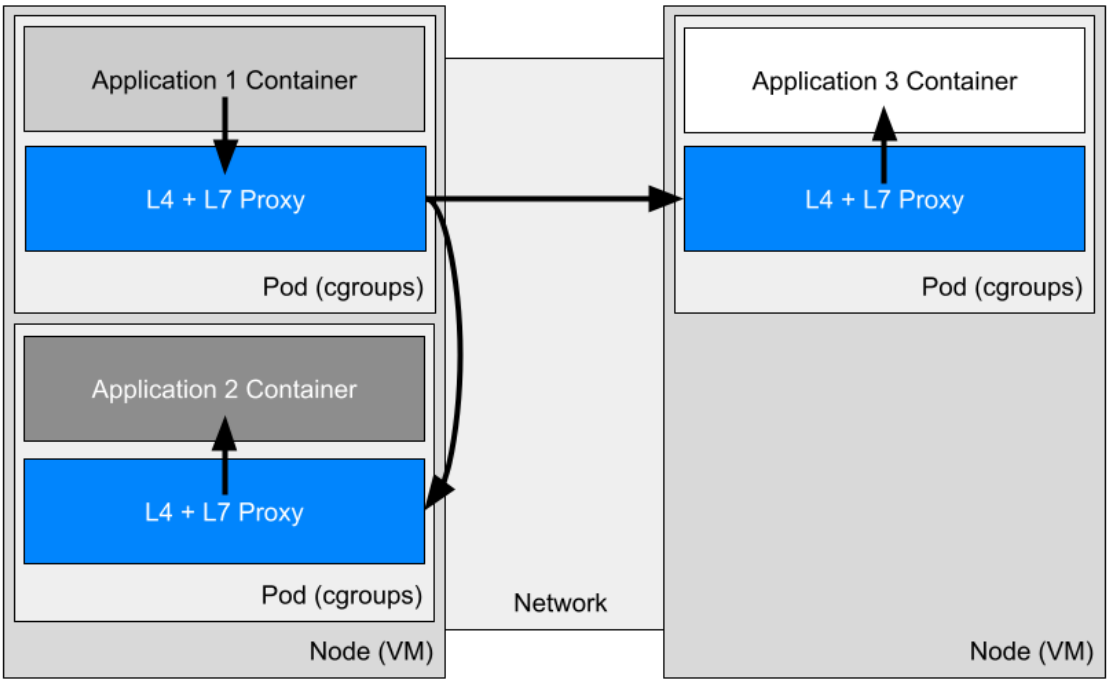


Fig. 1. Sidecar model —L4 and L7 Proxy per Service Instance (DPA-1)

(The combined L4 and L7 proxy is deployed for each application instance and there is no sharing of any proxy).

3.2. Shared L4 – L7 per Service Model (DPA-2)

In Shared L4 – L7 per Service architecture, there is a shared L4 proxy per node (i.e., shared among all service instances that execute on the same physical host) with L7 proxies dedicated per service account. A variation in this architecture is to dedicate an L7 proxy for an entire namespace. This is not desirable from a security viewpoint for the same reasons that shared service accounts for an entire namespace are not recommended [2]. Hence, it is not considered for threat analysis in this document. An example of the implementation of this data plane architecture is Istio's "ambient mode", where the per node L4 proxy is called a Ztunnel proxy, and the per service account L7 proxy is called a Waypoint proxy [6][7][10][11][13]. It is to be noted that not every service needs a L7 proxy but when a L7 proxy is installed, it is dedicated to a single service.

A schematic diagram of this architecture is shown in Fig. 2.

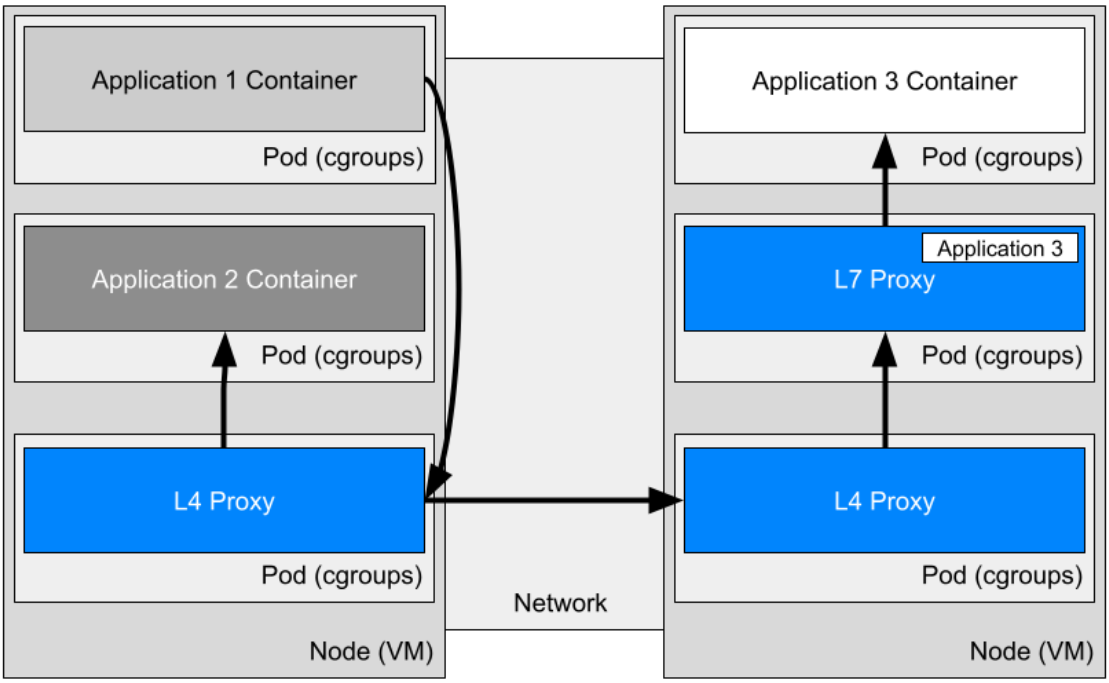


Fig. 2. Shared L4 – L7 per Service Model (DPA-2)

(An L4 proxy is deployed on each node and shared by all applications on that node. A single L7 proxy instance is deployed on behalf of "Application 3").

3.3. Shared L4 and L7 Model (DPA-3)

In Shared L4 and L7 architecture, the L4 and L7 functions are implemented on a per node basis. There is a shared L7 proxy per node, i.e., shared among all service instances that execute on the same physical host and provides L7 functions for all services in that node. L4 functions can be implemented by the L7 data plane, like the sidecar model (DPA-1) but shared across all applications on the node rather than dedicated one-per-app-instance. However, in some instances, L4 functions (e.g., traffic routing) can be performed by in-kernel programs (e.g., Extended Berkeley Packet Filter (eBPF) programs [8]) instead of by a mesh proxy. An example of this data plane architecture is the Cilium service mesh, which deploys the Envoy proxy as an L7 proxy based on its CiliumEnvoyConfig specification [8][9][12].

A schematic diagram of this architecture is shown in Fig. 3.

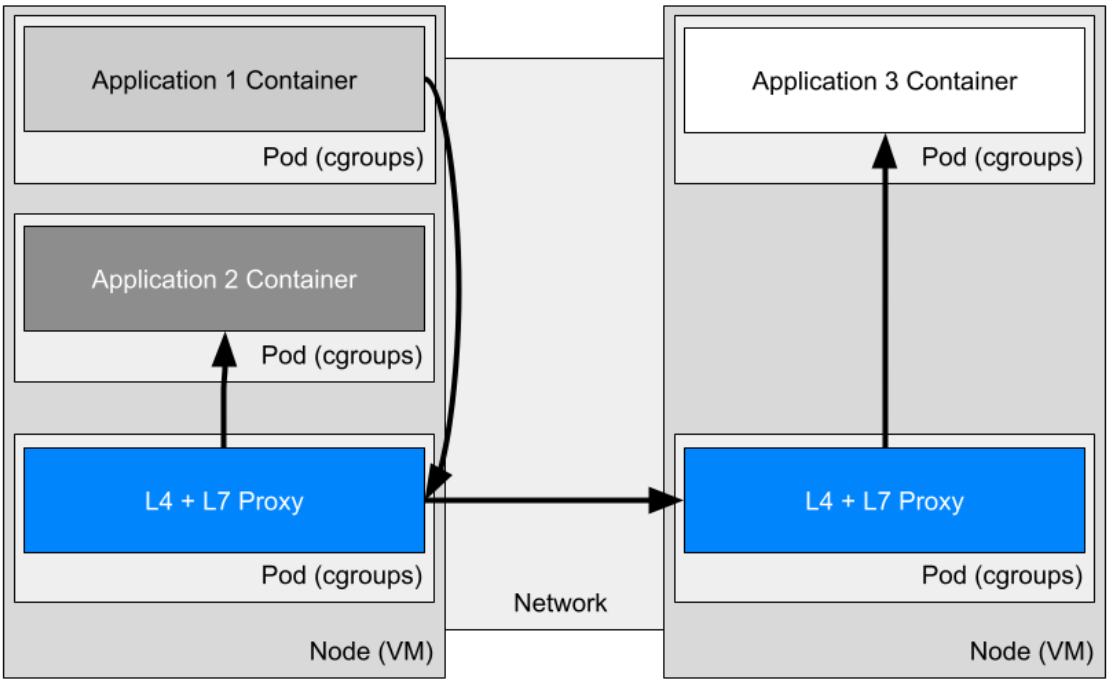


Fig. 3. Shared L4 – L7 Model (DPA-3)

3.4. L4 and L7 as Part of the Application Model (DPA-4)

L4 and L7 Part of the Application Model is a data plane architecture that does not have any pre-installed proxies. Instead, the service mesh control plane dynamically configures access to proxy functions using a set of discovery APIs collectively known as xDS APIs. The gRPC client and server libraries for applications provides extensive support for the xDS APIs. Leveraging this feature, the service mesh control plane can program L4 and L7 functions into this library in the service container. These gRPC libraries can then provide L4 and L7 functionality (in general, all policy enforcements) to the workloads or service instances with which they are integrated, thus replicating the exact services that the L4 and L7 proxies provide to those workloads [14].

A schematic diagram of this architecture is shown in Fig. 4.

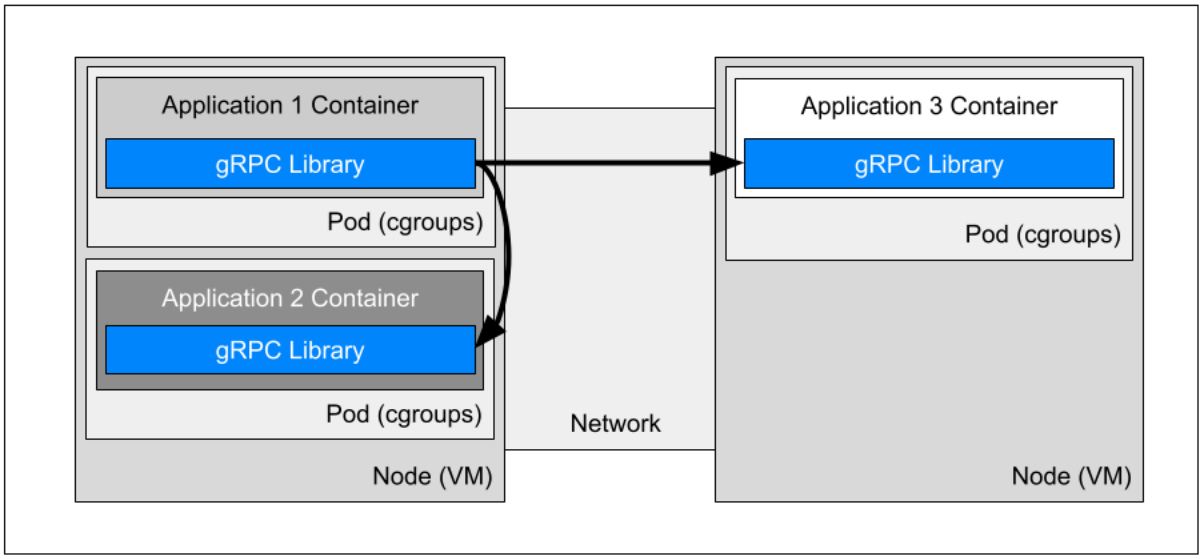


Fig. 4. L4 and L7 as Part of the Application Model (gRPC proxyless Model) (DPA-4)

4. Data Plane Architecture Threat Scenarios and Analysis Methodology

The threat scenario considers a service mesh deployed in a Kubernetes cluster with the assumption that no human can directly access the cluster, achieved via k8s role-based access control (RBAC). The only interaction with the cluster is via a continuous integration/continuous delivery (CI/CD) system-controlled declarative configuration in a version-controlled repository with a multi-step approval process to change that configuration, including the approval of each change by one other human at minimum.

This scenario starts by identifying access by external threat actors, internal threat actors, and malicious co-tenants.

External threat actors include:

- Compromised workload (application) container (e.g., via a supply chain attack)
- Compromised node L4 proxy or container network interface (CNI)
- Compromised node L7 proxy
- Compromised node with limited privileged access (e.g., a container breakout)
- Root compromise of node (e.g., a container breakout chained with the exploitation of a privilege escalation vulnerability)
- Network access to the Kubernetes API server

Internal threat actors include:

- Cluster administrators who have wide-ranging rights to view the cluster and approve changes to the version-controlled repository and may even have direct access to the Kubernetes cluster (e.g., via a break-glass debugging account; such super-accounts should generate detailed audit records of their usage)
- Application developers who can build images and approve configurations that go into the cluster.
- Infrastructure engineers who have permission to deploy and configure the mesh gated by the version-controlled repository's approval process.
- Compromised network infrastructure between nodes (e.g., unencrypted cross-data center communication)

Finally, malicious co-tenants - In general, k8s is not a hard multi-tenant system (i.e., one where tenants do not trust each other), and isolating tenants from each other with stronger boundaries is recommended. In this context, a malicious co-tenant would fall into one of the internal threat actor categories above. In this context, consider the following threats in an environment as they relate to the service mesh:

1. Compromised L4 proxy
2. Compromise of the application container
3. Compromise of business data

4. Compromised L7 proxy
5. Compromised shared L7 Proxy
6. Outdated client libraries in applications
7. Denial of service (DoS)
8. Resource consumption
9. Privileged L4 Proxy
10. Bypassing traffic interception

4.1. Threat Analysis Methodology

The threat analysis methodology begins by identifying 10 potential threats to the components that make up the four architectural patterns for the proxy model or data plane architecture. The methodology then describes how the functionality of each component of the architecture is adversely affected by each threat and rates the impact and likelihood of their occurrence, justifying each rating. There are three values for the ratings: low, medium, and high. The values assigned to these ratings are relative to other data plane architectures and are not absolute values based on a metric. For example, assigning a rating value of "high" to the likelihood parameter for a threat does not imply that the threat is highly likely in all situations. Rather, it means that the threat is most likely to be executed against that architecture *relative to the other architectures under discussion*.

The *impact (I)* of the exploitation of each threat is then evaluated along with the *likelihood (L)* of that threat being exploited. For both parameters, there is a rating of low, medium, or high, which are translated to numeric scores of 1, 2, and 3, respectively. Multiplying *I * L* yields an indication of how important that threat is and the necessity to mitigate that threat relative to other architectures under discussion. Summing up the values of this indicator for all 10 potential threats provides an indication of the threat profile for that architectural pattern.

Threats whose impact and likelihood are the same irrespective of the architecture (i.e., the threats are agnostic to the architecture) are assigned a relative score of 1 for impact and 1 for likelihood.

5. Detailed Threat Analysis for Data Plane Architectures

This section considers potential targeted threats to L4 and L7 proxy functions or the libraries that implement the associated functions, the relevant proxy function that is impacted, the degree of impact, and the likelihood of the threat occurring for each of the data plane architectures (see Sec. 3.1 to 3.4).

5.1. Threat Analysis for L4 and L7 Proxy per Service Instance (DPA-1) — Sidecar Model

The threats to the data plane of the service mesh are denoted using the mnemonic TR-x, where TR stands for threat, and x stands for the threat sequence number.

5.1.1. Compromised L4 Proxy (TR-1)

Threat Description*:* A compromised L4 proxy (or L4 functions in the case of sidecar proxy with combined L4 and L7 functions) leads to leaked identities for every workload (service) running on the node.

Proxy Function Impacted*:* Sidecar proxies negotiate mTLS connections (for communicating with any other service) on behalf of the single workload with which it is associated. In order to compromise key material and identity documents (i.e., threat targets) for multiple workloads, multiple proxy (sidecar) instances would need to be compromised.

Impact Score = 1: Because of the nature of impact discussed above (i.e., single workload/single identity affected), this threat is assigned an impact score of 1.

Likelihood Score = 2*:* Code relating to L7 functions is present to be exploited if it can be triggered. In a pure L4 proxying case, it *should* not be triggerable, but this relies on correct configuration from users and the service mesh implementation.

5.1.2. Compromised Application Container (TR-2)

Threat Description*:* A compromised application container (e.g., via a supply chain attack during the development phase) leads to a takeover of the identity associated with that application.

Proxy Function Impacted*:* Proxies run in the same network space (i.e., same pod in Kubernetes environment) as the application container, meaning that a compromise of the application container that hosts the service instance can easily lead to a compromise of any key material (e.g., full access to key material pertaining to the identity of the service) possessed by the proxy.

Impact Score = 2: Because of the nature of impact discussed above (i.e., single workload/single identity affected), this threat is assigned an impact score of 2. Even though only a single identity is compromised, like TR-1, this has a higher impact score as the application itself must be updated. A compromised proxy can be remediated without requiring the application itself to be updated, so there is a higher chance that a central team can successfully remediate a compromise without involving application teams.

***Likelihood Score = 1**:* The likelihood score is the same, regardless of architecture.

5.1.3. Compromise of Business Data (TR-3)

***Threat Description**:* A compromised identity is used to pivot through the infrastructure in order to compromise the confidentiality, integrity, or availability of business data.

***Proxy Function Impacted and Impact Score (= 1) and Likelihood Score (= 1)**:* The scores are the same, regardless of architecture. This is the fundamental risk of identity-based policy and why the principle of least privilege (PoLP) should be practiced. The telemetry provided by the service mesh (regardless of architecture) is invaluable for understanding communications in a system and creating accurate access policies.

5.1.4. Compromised L7 Proxy (TR-4)

***Threat Description**:* There is a vulnerability in the L7 processing stack of the service mesh proxy. Since L7 processing is inherently more complex, there is a higher probability for vulnerabilities to arise in this part of the stack, as supported by historical Common Vulnerabilities and Exposures (CVE) data.

***Proxy Function Impacted**:* There is no separation between L4 and L7 processing. Although any exploitable vulnerability in a sidecar proxy can lead to the compromise of all identities in the mesh, this would involve more individual proxy instances being compromised, and it may be more difficult for an attacker to accomplish this feat undetected.

Impact Score = 1: A single workload is impacted (e.g., leaking credentials, becoming unavailable due to a denial-of-service attack). The same exploit could be used against all sidecars in the mesh with applications opting in to the L7 behavior, resulting in a compromise of all identities (Impact 3). In practice, this requires many more events than any other architecture and increases the likelihood of detecting and responding to the event in a timely manner.

***Likelihood Score = 1**:* A full L7 capability is available in the proxy, meaning that a relatively large attack surface is exposed. In practice for the service mesh use case, however, the HTTP processing is most likely targeted. If the application is using L7 mesh capabilities, they would be vulnerable to exploit.

5.1.5. Compromise of Shared L7 Proxy (TR-5)

***Threat Description**:* A co-tenant exploits an L7 traffic processing vulnerability in a shared proxy, which affects the confidentiality, integrity, or availability of traffic to and from another workload running on the same node.

***Proxy Function Impacted**:* Because the proxy is dedicated per application, the impact on availability is limited to the resource constraints imposed by the scheduling system (e.g., Kubernetes). Confidentiality is impacted the same as if another application itself is compromised (i.e., containers provide some guarantee; micro-VMs provide a stronger degree of isolation; full-blown VMs are the strongest).

Impact Score = 1: For noisy neighbors (i.e., other L7 proxies on the same host that are compromised), the impact is limited by the underlying scheduling and resource constraint system (e.g., k8s, VM sizing). The following are identical across all architectures: for a shared ingress gateway, all services exposed on that gateway would be impacted (Impact 2); for a shared egress gateway, all services that utilize the egress gateway would be impacted (Impact 3). Typically, only a single deployment of egress gateways is used.

Likelihood Score = 1*:* The sidecar itself is not a shared proxy. By its nature, it is dedicated to an individual application. In this case, TR-5 refers to noisy neighbors, other proxies on the same node that cause a denial of service (DoS), and shared ingress or egress gateways. Noisy neighbors are mitigated based on the degree of isolation of the host (i.e., container versus micro-VM versus VM). The likelihood of exploiting a shared L7 ingress or egress gateway is the same across all architectures.

5.1.6. Outdated Client Libraries in Applications (TR-6)

Threat Description*:* Client libraries are not updated frequently or consistently across the estate of microservices, leading to potential vulnerabilities and weaknesses that can be exploited.

Proxy Function Impacted*:* The proxy's infrastructure code is decoupled from its application code.

Impact Score = 1: The mesh infrastructure is separate from the application itself and, therefore, not directly impacted by application vulnerabilities. Instead, a compromised application would use the functioning mesh to hijack the application's identity. Some application vulnerabilities can be mitigated via policies enforced by the mesh (e.g., mesh-enforced web application firewall (WAF) policy can help mitigate an application vulnerability like Log4j while the organization is patching applications).

Likelihood Score = 1*:* The likelihood is the same, regardless of architecture.

5.1.7. Denial of Service (TR-7)

Threat Description*:* There is a conventional DoS threat.

Proxy Function Impacted*:* Because the proxy is a per app instance, a DoS needs to be executed per app. Because the proxy shares resources with the application, a DoS on the mesh data plane directly competes for resources with the app instance itself. The overall blast radius of the DoS is as strong as the underlying isolation mechanism that protects workloads (i.e., pods) from each other (e.g., VMs, micro-VMs, containers).

Impact Score = 1: There is a single instance of a single application.

Likelihood Score = 1*:* L4 and L7 code may be exploited. However, the attack must be executed across each instance of the target. There is not a central resource to target to achieve a DoS other than a shared ingress gateway that is identical across all architectures under discussion.

5.1.8. Resource Consumption (TR-8)

***Threat Description**:* There is overall resource consumption by the data plane of the service mesh infrastructure.

***Proxy Function Impacted**:* Because sidecars are a separate process and dedicated per application, they have the worst overall resource consumption. Configuration that is identical across all applications must be held by the data plane per application and cannot be shared. The static overhead of the sidecar data plane implementation itself (e.g., constant Random Access Memory (RAM) usage, constant Central Processing Unit (CPU) overhead) is duplicated per application instance and cannot be amortized over all applications on the node. In part, this isolation is what allows sidecars to have lower impact and likelihood across many of the other threats identified here.

Impact Score = 3: Sidecars will consume the most resources out of all available options, even in well-configured environments.

***Likelihood Score = 3**:* It is challenging to configure sidecars correctly to minimize configuration and reduce overhead. While some specific implementations do better than others due to engineer trade-offs (e.g., lazily loading configuration the first time that an application needs it versus eagerly pushing all configuration ahead of use), it is a more common situation where the most resource utilization occurs with a sidecar architecture.

5.1.9. Privileged L4 Proxy (TR-9)

***Threat Description**:* A service mesh implementation requires an L4 component (e.g., deployed as a DaemonSet on a Kubernetes cluster) to run with an overprivileged security context (e.g., Privileged Pod).

***Proxy Function Impacted & Impact Score (= 1) and Likelihood Score (= 1)**:* The impact and likelihood are the same, regardless of architecture. In a per-node case, this is usually encapsulated as a CNI provider that runs in a privileged context by default. In a sidecar case, privilege is only needed at startup to establish traffic interception rules. Depending on the implementation (e.g., Kubernetes init containers), this can ensure that the privileged user is not run alongside the application but only during initialization. In all cases, Kubernetes-defined CAP_NET_ADMIN is typically the only privilege required for mesh data plane functionality.

5.1.10. Data Plane (Service Mesh) Bypassed (TR-10)

***Threat Description**:* Traffic is sent directly to a workload, bypassing mesh functionality and authorization policies.

***Proxy Function Impacted**:* It is the easiest to bypass of all of the available models, from the application choosing not to use a sidecar to the container-local bypasses/configurations.

Impact Score = 2: An application is exposed without mesh security controls.

Likelihood Score = 2*:* Because the proxy runs in user space in the same control groups as the application, there are a variety of attacks available that are not relevant or applicable to other implementations.

5.1.11 Overall Threat Score

Cumulative Threat Score = 23*:* This is computed based on the methodology in Sec. 4.1.

5.2. Threat Analysis for Shared L4 – L7 per Service Model (DPA-2)

5.2.1. Compromised L4 Proxy (TR-1)

Threat Description*:* A compromised L4 proxy (or L4 functions in the case of sidecar proxy with combined L4 and L7 functions) leads to leaked identities for every workload (i.e., service) running on the node.

Proxy Function Impacted*:* The L4 proxy has access to all of the keys associated with the workloads running on the node.

Impact Score = 3*:* The identities of all of the workloads (i.e., services) on the node are compromised.

Likelihood Score = 1*:* Only code that delivers L4 functions is present. This minimal code footprint and functionality present the lowest attack surface of all options.

5.2.2. Compromised Application Container (TR-2)

Threat Description*:* A compromised application container (e.g., via a supply chain attack during the development phase) leads to a takeover of the identity associated with that application.

Proxy Function Impacted*:* Data plane components are not located in the same pod as workload containers, so a compromised workload does not necessarily lead to the access of keys or secrets.

Impact Score = 1: A single workload/single identity is impacted, and there is no direct access to underlying key material.

Likelihood Score = 2*:* The likelihood is the same, regardless of architecture.

5.2.3. Compromise of Business Data (TR-3)

Threat Description*:* An identity is used to pivot through the infrastructure in order to compromise the confidentiality, integrity, or availability of business data.

Proxy Function Impacted and Impact Score (= 1) and Likelihood Score (= 1)*:* The impact and likelihood are the same, regardless of architecture. This is the fundamental risk of identity-based policy and why the PoLP should be practiced. The telemetry provided by the service

mesh (regardless of architecture) is invaluable for understanding communication patterns in a given system and creating accurate access policies.

5.2.4. Compromised L7 Proxy (TR-4)

***Threat Description**:* There is a vulnerability in the L7 processing stack of the service mesh proxy. As L7 processing is inherently more complex, there is a higher probability for vulnerabilities to arise in this part of the stack, as supported by historical CVE data.

***Proxy Function Impacted**:* This topology allows for "less complex" L4 capabilities (e.g., mTLS to be adopted with L7 processing only occurring if there is a strict requirement for it). Each service account has its own dedicated L7 proxy.

Impact Score = 2: A single set of workloads is impacted (e.g., DoS) or a single identity leaked. In the event of a DoS, it is much easier to make all workloads unavailable compared to the sidecar model because the mesh's L7 processing is centralized into L7 "middle proxies."

***Likelihood Score = 1**:* The likelihood is the same as sidecar (i.e., potential impact on all workloads using L7 capabilities) (see Sec. 5.1.4).

5.2.5. Compromise of Shared L7 Proxy (TR-5)

***Threat Description**:* A co-tenant exploits an L7 traffic processing vulnerability in a shared proxy and affects the confidentiality, integrity, or availability of traffic to and from another workload that runs on the same node.

***Proxy Function Impacted**:* By limiting the per-node functionality to L4 processing, the attack surface is significantly reduced.

Impact Score = 1: The application workload itself is unaffected. The proxy is a separate deployment. As long as the L7 proxy is not shared with the compromised application, there is no impact.

***Likelihood Score = 1**:* The likelihood is as likely as in TR-4 (see Sec. 5.2.4).

5.2.6. Outdated Client Libraries in Applications (TR-6)

***Threat Description**:* Client libraries are not updated frequently or consistently across the estate of microservices, leading to potential vulnerabilities and weaknesses that can be exploited.

***Proxy Function Impacted**:* The infrastructure code is decoupled from the application code.

Impact Score = 1: The impact is the same as in DPA-1 (see Sec. 5.1.6).

***Likelihood Score = 1**:* The likelihood is the same, regardless of architecture.

5.2.7. Denial of Service (TR-7)

***Threat Description**:* There is a conventional DoS threat.

***Proxy Function Impacted**:* A DoS executed at L4 has the same impact as the centralized per-node model because the L4 process is centralized per node. All applications on the node are impacted. A DoS executed at L7 impacts all application instances of the target application since a set of dedicated L7 proxies is deployed per app. The number of proxies that implement L7 functionality is typically less than the number of application instances, making them an easier target for DoS than every instance of the target application.

Impact Score = 2: An L4 DoS would impact all application instances on the target host.

***Likelihood Score = 2**:* The L4 proxy is deployed once per node, so it presents a better target for DoS than DPA-1 or DPA-4. This is mitigated somewhat by the simplified functionality of an L4 proxy compared to a combined L4 and L7 proxy. The L7 proxy is shared by multiple instances of the same application and presents an easier DoS target than the application itself. Therefore, it is more likely than the sidecar model (DPA-1).

5.2.8. Resource Consumption (TR-8)

***Threat Description**:* There is overall resource consumption by the data plane of the service mesh infrastructure.

***Proxy Function Impacted**:* The shared L4 proxy typically has a much lower memory (RAM) footprint and CPU usage due to a lower rate of change of configuration, less configuration overall, and less responsibility than a combined L4 and L7 sidecar proxy (DPA-1). For the service mesh's data plane, L7 processing is typically the dominating CPU cost, followed by encryption. L7 proxies are shared by all instances of the same application and deployed as a few traditional "reverse proxies" per application. This results in much lower resource consumption for L7 processing than the sidecar model (DPA-1). Overall, DPA-2 uses more resources than the shared per node model (DPA-3) but substantially less than the sidecar model (DPA-1) due to reduced overhead (e.g., an application with 50 instances requires 50 sidecars but might be served with five shared L7 proxies or fewer).

Impact Score = 2: DPA-3 has lower consumption than sidecar and is easier to achieve but is not as low as all shared (DPA-3) or all in app (DPA-4).

***Likelihood Score = 1**:* This model was intentionally designed to mitigate the resource overhead of running an entire sidecar proxy instance per service instance. Instead, the single node-level proxy receives full configurations for all services on the node, and dedicated L7 proxies receive configuration for only their service. As a result, total resource consumption is low by default.

5.2.9 Privileged L4 Proxy (TR-9)

***Threat Description**:* A service mesh implementation requires an L4 component (e.g., deployed as a DaemonSet on a Kubernetes cluster) to run with an overprivileged security context (e.g., Privileged Pod).

***Proxy Function Impacted and Impact Score (= 1) and Likelihood Score (= 1)**:* The impact and likelihood are the same, regardless of architecture. In the per-node case, this is usually

encapsulated as a CNI provider that runs in a privileged context by default. In a sidecar case, privilege is only needed at startup to establish traffic interception rules. Depending on the implementation (e.g., Kubernetes init containers), this can ensure that the privileged user is not run alongside the application but only during initialization. In all cases, Kubernetes-defined CAP_NET_ADMIN is typically the only privilege required for mesh data plane functionality.

5.2.10 Data Plane (Service Mesh) Bypassed (TR-10)

***Threat Description**:* Traffic is sent directly to a workload, bypassing mesh functionality and authorization policies.

***Proxy Function Impacted**:* Part of the goal of moving enforcement out of the application context and into a shared context is to use stronger primitives to ensure the non-bypassability of the mesh data plane. In general, with a per-node L4 setup, sending traffic to an individual application instance on the node should not be achievable (e.g., similar to but not necessarily implemented as a host-level Virtual Private Network (VPN) that requires workloads to be part of the VPN overlay to connect). L7 proxies are deployed independently from the applications that they represent, which requires special configuration in the mesh to ensure that they are routed through and making the ability to bypass easier than other models. The impact of a missing L7 policy can be significant. Other models rely on one thing at most to ensure that traffic is directed to the correct policy enforcement point. This model relies on two things to ensure that traffic is subject to the correct policy enforcement points (PEPs): traffic interception and mesh configuration to route via middle proxies.

Impact Score = 2: An app is exposed without mesh security controls.

***Likelihood Score = 2**:* L4 controls are designed and built to mitigate this. L7 controls are easier to bypass compared to the sidecar model.

5.2.11 Overall Threat Score

Cumulative Threat Score = 22: This is computed based on the methodology in Sec. 4.1.

5.3. Threat Analysis for Shared L4 and L7 Model (DPA-3)

5.3.1. Compromised L4 Proxy (TR-1)

***Threat Description**:* A compromised L4 proxy (or L4 functions in the case of sidecar proxy with combined L4 and L7 functions) leads to leaked identities for every workload (i.e., service) running on the node.

***Proxy Function Impacted**:* The L4 proxy has access to all of the keys associated with the workloads running on the node.

***Impact Score = 3**:* All identities on the node are impacted.

Likelihood Score = 3*:* L7 code may be enabled for another server that can be exploited to affect all applications on the host.

5.3.2. Compromised Application Container (TR-2)

Threat Description*:* A compromised application container (e.g., via a supply chain attack during the development phase) leads to a takeover of the identity associated with that application.

Proxy Function Impacted*:* Data plane components are not located in the same pod as workload containers, so a compromised workload does not necessarily lead to the access of keys or secrets.

Impact Score = 1: A single workload/single identity is impacted, and there is no direct access to underlying key material.

Likelihood Score = 2*:* The likelihood is the same, regardless of architecture.

5.3.3. Compromise of Business Data (TR-3)

Threat Description*:* An identity is used to pivot through the infrastructure in order to compromise the confidentiality, integrity, or availability of business data.

Proxy Function Impacted and Impact Score (= 1) and Likelihood Score (= 1)*:* The impact and likelihood are the same, regardless of architecture. This is the fundamental risk of identity-based policy and is why the PoLP should be practiced. The telemetry provided by the service mesh (regardless of architecture) is invaluable for understanding communication patterns in a given system and creating accurate access policies.

5.3.4. Compromised L7 Proxy (TR-4)

Threat Description*:* There is a vulnerability in the L7 processing stack of the service mesh proxy. As L7 processing is inherently more complex, there is a higher probability for vulnerabilities to arise in this part of the stack, as supported by historical CVE data.

Proxy Function Impacted*:* This topology allows for "less complex" L4 capabilities (e.g., mTLS to be adopted with L7 processing only occurring if there is a strict requirement for it). The blast radius of a proxy compromise affects all workloads on the node. This means that its failure represents a shared fate outage and is susceptible to DoS attacks.

Impact Score = 3: An L7 capability is shared across all applications on the node, so a single application's configuration may cause the proxy to become susceptible to failure and result in attacks on all applications on the node (e.g., a credential leak or denial of service, depending on the attack).

Likelihood Score = 2*:* For a given application using L7 capabilities, the likelihood is the same as for the sidecar model. However, because workloads that are only doing L4 are susceptible to attack, the likelihood is higher if they share the same node that would have been safe under the sidecar model (DPA-1).

5.3.5. Compromise of Shared L7 Proxy (TR-5)

***Threat Description**:* A co-tenant exploits an L7 traffic processing vulnerability in a shared proxy and affects the confidentiality, integrity, or availability of traffic to or from another workload that runs on the same node.

***Proxy Function Impacted**:* A single proxy instance does not provide an inherently multi-tenant setup. Hence, security concerns arise when combining complex processing rules for L7 traffic from multiple unconstrained tenants in a shared instance. In this configuration, L7 processing of the traffic of multiple co-tenants is performed within one process with no memory protection or isolation benefits that could be gained by containerizing L7 functionality per workload.

Impact Score = 3: All workloads on the node are impacted.

***Likelihood Score = 2**:* A compromise is as likely as in the sidecar model (DPA-1), but applications that would not be susceptible to attack under DPA-1 *are* susceptible under this model (DPA-3) (see Sec. 5.3.4).

5.3.6. Outdated Client Libraries in Applications (TR-6)

***Threat Description**:* Client libraries are not updated frequently or consistently across the estate of microservices, leading to potential vulnerabilities and weaknesses that can be exploited.

***Proxy Function Impacted**:* The infrastructure code is decoupled from the application code.

Impact Score = 1: The impact is the same as the sidecar model (DPA-1) (see Sec. 5.1.6).

***Likelihood Score = 1**:* The likelihood is the same, regardless of architecture.

5.3.7. Denial of Service (TR-7)

***Threat Description**:* There is a conventional DoS threat.

***Proxy Function Impacted**:* Because processing for all application instances on the node is shared and a single proxy instance is not inherently multi-tenant (i.e., provides no controls with regard to resource utilization across independent backends and clients), the blast radius of a DoS threat on the mesh data plane affects every application on the node.

Impact Score = 3: The impact applies to all workloads on the node.

Likelihood Score = 2: If *any* application configuration triggers exploitable paths in the shared proxy, *all* applications on the node suffer.

5.3.8. Resource Consumption (TR-8)

***Threat Description**:* There is overall resource consumption by the data plane of the service mesh infrastructure.

***Proxy Function Impacted**:* Because *all* functionality is shared at the node level, DPA-3 has the most opportunity for deduplication and a subsequent reduction in resource usage. A

configuration (e.g., service discovery) needs to be sent only once to each node rather than to each and every application instance. Overall, this results in the lowest rate of change, the least data transferred, and a lower runtime footprint (e.g., RAM, CPU).[1]

Impact Score = 1: This has the lowest overall resource utilization of all available architectures.

Likelihood Score = 1*:* Because there is only a single proxy instance per node, rather than a data plane instance per service or per service instance, resource consumption should be the lowest of all out-of-process (i.e., non-gRPC) models.

5.3.9. Privileged L4 Proxy (TR-9)

Threat Description*:* A service mesh implementation requires an L4 component (e.g., deployed as a Daemon Set on a Kubernetes cluster) to run with an overprivileged security context (e.g., Privileged Pod).

Proxy Function Impacted and Impact Score (= 1) and Likelihood Score (= 1)*:* The impact and likelihood are the same, regardless of architecture. In the per-node case, this is usually encapsulated as a CNI provider that runs in a privileged context by default. In the sidecar case, privilege is only needed at startup to establish traffic interception rules. Depending on the implementation (e.g., Kubernetes init containers), this can ensure that the privileged user is not run alongside the application but only during initialization. In all cases, Kubernetes-defined CAP_NET_ADMIN is typically the only privilege required for mesh data plane functionality.

5.3.10. Data Plane (Service Mesh) Bypassed (TR-10)

Threat Description*:* Traffic is sent directly to a workload, bypassing mesh functionality and authorization policies.

Proxy Function Impacted*:* Part of the goal of moving enforcement out of the application context and into a shared context is to use stronger primitives to ensure the non-bypassability of the mesh data plane. In general, with a per-node setup, sending traffic to an individual application instance on the node should not be achievable (e.g., similar to but not necessarily implemented as a host-level VPN that requires workloads to be part of the VPN overlay to connect).

Impact Score = 3: All applications on the node are exposed without mesh security controls.

Likelihood Score = 1*:* By design, this architecture is built to mitigate this kind of bypass.

5.3.11 Overall Threat Score

Cumulative Threat Score = 37*:* This is computed based on the methodology in Sec. 4.1.

[1] Some implementations do not fully de-dupe configurations (e.g., due to implementation, as a security measure to provide some degree of isolation) and consume RAM more similarly to a sidecar case than might otherwise appear.

5.4. Threat Analysis for L4 and L7 as Part of the Application Model (gRPC proxyless Model (DPA-4))

5.4.1. Compromised L4 Proxy (TR-1)

***Threat Description**:* A compromised L4 proxy (or L4 functions in the case of sidecar proxy with combined L4 and L7 functions) leads to leaked identities (cannot it not lead to MITM attacks by compromising proxy logic ?) for every workload (i.e., service) that runs on the node.

***Proxy Function Impacted**:* mTLS connections are negotiated by the client library inside of the application with a single identity (i.e., that of the application). In order to compromise key material and identity documents for multiple workloads, multiple application instances would need to be compromised.

***Impact Score = 1**:* A single workload/single identity is impacted.

***Likelihood Score = 2**:* There is a large surface area if something goes wrong since this is inside of the application's context. Therefore, this is as likely or slightly more likely than DPA-1.

5.4.2. Compromised Application Container (TR-2)

***Threat Description**:* A compromised application container (e.g., via a supply chain attack during the development phase) leads to the takeover of an identity associated with that application.

***Proxy Function Impacted**:* Compromising the application *does* compromise the mesh. Full access to any key material used by the application, including the mesh identity, is achievable.

Impact Score = 2: A single workload/single identity is impacted, and there is full access to the key material used by that application.

***Likelihood Score = 2**:* The likelihood is the same, regardless of architecture.

5.4.3. Compromise of Business Data (TR-3)

***Threat Description**:* An identity is used to pivot through the infrastructure in order to compromise the confidentiality, integrity, or availability of business data.

***Proxy Function Impacted and Impact Score (= 1) and Likelihood Score (= 1)**:* The impact and likelihood are the same, regardless of architecture. This is the fundamental risk of identity-based policy and is why the PoLP should be practiced. The telemetry provided by the service mesh (regardless of architecture) is invaluable for understanding communication patterns in a given system and creating accurate access policies.

5.4.4. Compromised L7 Proxy (TR-4)

***Threat Description**:* There is a vulnerability in the L7 processing stack of the service mesh proxy. As L7 processing is inherently more complex than typical L4 proxy processing, there is a higher

probability for vulnerabilities to arise in this part of the stack, as supported by historical CVE data.

***Proxy Function Impacted**:* Compromising the L7 processing stack results in compromising the entire application and in more risk of compromise beyond runtime identity and DoS for other users.

Impact Score = 3: The application itself is compromised, including non-mesh credentials (e.g., truncate table users) that are not available if only the proxy is compromised.

***Likelihood Score = 3**:* Since L7 processing code *is* the application, the surface area is much larger.

5.4.5. Compromise of Shared L7 Proxy (TR-5)

***Threat Description**:* A co-tenant exploits an L7 traffic processing vulnerability in a shared proxy and affects the confidentiality, integrity, or availability of traffic to or from another workload that runs on the same node.

***Proxy Function Impacted**:* L7 processing is entirely isolated by whatever mechanisms isolate the applications themselves (e.g., containers, micro-VMs, VMs). The impact is limited by the strength of that boundary.

Impact Score = 1: See Sec. 5.1.5.

***Likelihood Score = 1**:* The likelihood is the same as any other application compromise.

5.4.6. Outdated Client Libraries in Applications (TR-6)

***Threat Description**:* Client libraries are not updated frequently or consistently across the estate of microservices, leading to potential vulnerabilities and weaknesses that can be exploited.

***Proxy Function Impacted**:* Infrastructure concerns are embedded within the application code. Challenges can arise when enforcing consistency in versions between microservices.

Impact Score = 3: The mesh functionality itself is part of the application. Therefore, bad application updates result in bad mesh updates. This means that vulnerabilities remain for longer, and since the mesh is part of the application, a vulnerability in the application *is* a vulnerability in the mesh data plane.

***Likelihood Score = 2**:* The likelihood depends on the frequency of updates. If applications can be updated quickly (i.e., on the order of minutes to hours), the likelihood is low. If applications take weeks or months to be updated, the likelihood is high. Cross-cutting concerns (e.g., the mesh data plane, which is critical to the organization's overall security posture) should be patched as soon as possible.

5.4.7. Denial of Service (TR-7)

***Threat Description**:* There is a conventional denial-of-service threat.

***Proxy Function Impacted**:* A DoS threat to the mesh data plane (L4 *or* L7) is a threat to the application itself. In all other respects, it is very similar to the sidecar model.

Impact Score = 1: There is a single instance of a single application, so an attack could be repeated across all applications (see Sec. 5.1.7).

***Likelihood Score = 2**:* The functionality of both the mesh data plane and the application code itself are susceptible to DoS.

5.4.8. Resource Consumption (TR-8)

***Threat Description**:* There is overall resource consumption by the data plane of the service mesh infrastructure.

***Proxy Function Impacted**:* Because it is built into the application, resources devoted to mesh data plane functionality are *very* low. The only reason that overall resource utilization may be higher than the shared L4 and L7 model (DPA-3) is because some duplication of configuration and processing needs to occur to push the configuration to every application instance.

Impact Score = 2: This has a potentially lower resource usage on a per-app basis than any other model but is likely higher in aggregate because resources and configuration cannot be shared across data plane instances.

***Likelihood Score = 1**:* gRPC achieves the lowest overhead of all possible options because there is no separate data plane, strictly the application itself. If an application is already using gRPC, it will see little-to-no additional overhead for using a mesh data plane to configure gRPC functionality.

5.4.9 Privileged L4 Proxy (TR-9)

***Threat Description**:* A service mesh implementation requires an L4 component (e.g., deployed as a Daemon Set on a Kubernetes cluster) to run with an overprivileged security context (e.g., Privileged Pod).

***Proxy Function Impacted and Impact Score (= 0) and Likelihood Score (= 0)**:* The mesh data plane functionality runs in the application context without any special privileges and is the same as the application itself. No special capabilities or permissions are required to intercept traffic or implement policy enforcement.

5.4.10 Data Plane (Service Mesh) Bypassed (TR-10)

***Threat Description**:* Traffic is sent directly to a workload, bypassing mesh functionality and authorization policies.

***Proxy Function Impacted**:* The application *is* the enforcement point, so there is no bypassing.

Impact Score = 1: The application is exposed in a degraded state or without some controls.

***Likelihood Score = 1**:* Due to the nature of RPC frameworks and in-process enforcement, mesh data plane policy should not be able to be bypassed.

5.4.11 Overall Threat Score

***Cumulative Threat Score = 28**:* This is computed based on the methodology in Sec 4.1.

6. Recommendations Based on the Application Security Risk Profile

While the ratings or scores for the impact and likelihood parameters for different threats in different data plane architectures are dictated by the number of service instances affected, the risk profiles associated with the applications are determined by the criticality of the entire application with respect to the business process that it supports.

When developing threat profiles for architectural patterns (see Sec. 5), consider that the impact and likelihood parameters for some threats are the same irrespective of the proxy model or data plane architecture. Since the ratings assigned to these parameters are relative, both parameters are assigned a rating of 1, which results in an overall threat rating of 1 for those threats. The threats that fall under this category and should be ignored are:

- Compromise of business data (TR-3)
- Privileged L4 proxy (TR-9)

Additionally, threats that have no direct security implications but may have performance implications should also be ignored, namely Resource consumption (TR-8).

6.1 Cloud-Native Applications With Low Risk Profiles

The service mesh capability requirements for this class of application are:

- **LOW-REQ1:** Service-to-service authorization (i.e., Service A can call Service B) is based on network location/parameter (e.g., subnet) and authorization at the granularity of the called service method. Calling the user and a per-call request are not required.
- **LOW-REQ2:** Logging and metrics need to be captured only at the level of network parameters (e.g., source/destination IP address) and not at the level of per-call request.
- **LOW-REQ3:** All traffic management capabilities (e.g., load balancing, rate limiting) need to be enforced at the network connection level and not at the per-call request.

These capabilities essentially involve network-transport/network-level data that can be provided by the proxy's L4 functions. Hence, the following are recommendations for this class of application:

- Since all requirements can be met by L4 proxies or L4 functions built into the libraries, all four data plane architectures can be theoretically used.
- Since neither method-level nor per-call request handling is required, thus eliminating all L7 functions, data plane architectures that deploy an L7 proxy per service instance (i.e., side-car model [DPA-1]) expose an unnecessary attack surface. Therefore, either of the two models with a shared L4 proxy (i.e., DPA-2 or DPA-3) is recommended. A gRPC proxy-less model (i.e., DPA-4) is also usable for this class of applications, though it does expose a larger attack surface than DPA-2 or DPA-3.

6.2 Cloud-Native Applications with Medium Risk Profiles

The service mesh capability requirements for this class of application are:

- **MEDIUM-REQ1:** In addition to service-to-service authorization at the level of service, a full authorization policy at the method level (i.e., Service A can execute GET on B's billing method with valid end user credentials that contain the READ scope) is required.
- **MEDIUM-REQ2:** Logging and metrics data need to be captured at the level of network parameters (e.g., source/destination IP address) along with some metadata (e.g., the called service and method).
- **MEDIUM-REQ3:** All traffic management capabilities (e.g., load balancing, rate limiting) can be enforced at the network-connection level (as in low risk profile case) and not at the per-call request or per-method level.

These capabilities essentially involve both the network-transport/network-level data (all L4 functions) and some L7 functions (not all), such as authenticating user identities not only locally from tokens (e.g., JSON Web Tokens [JWT]) and remotely using standardized protocols (e.g., OAuth and OIDC). Since the use of L7 proxies with some limited functionality is mandatory, the following are recommendations for this class of application:

- Just like for applications with a low risk profile, all four data plane architectures can be theoretically used.
- Since L7 functions are limited, it is not essential to dedicate an L7 proxy for each service. Hence, data plane architectures that deploy an L7 proxy for each service (e.g., sidecar model [DPA-1]) may consume more resources than other models for limited additional assurance. In contrast, most exploitable vulnerabilities lie in L7 code. A shared L4 and L7 model (i.e., DPA-3) is not desirable since the shared L7 component introduces risks for all services that share the same physical host. Therefore, the shared L4 and L7 per service model (i.e., DPA-2) is likely the best combination of resource utilization and risk. A gRPC proxy-less model (i.e., DPA-4) that includes libraries for L4 functions and limited L7 functions is also recommended with similar risks but even less resource utilization than DPA-2 in most cases.

6.3 Cloud-Native Applications With High Risk Profiles

The service mesh capability requirements for this class of application are:

- **HIGH-REQ1:** A full user-to-resource-level access control is required in addition to (a) service-to-service authorization at the level of service and (b) a full authorization policy at the method level (i.e., Service A can execute GET on B's billing method with valid end user credentials that contain the READ scope). This necessitates the proxy making an external authorization call for each request.
- **HIGH-REQ2:** Logging and metrics meta data relating to a request must be captured (e.g., rate of requests, rate of positive outcomes, processing time for each request).

- **HIGH-REQ3:** All traffic management capabilities are required at the request level and should involve application layer parameters in addition to those at the network connection level.

These capabilities reveal that a complete suite of L7 functions is required:

- Just like for applications with low risk and medium risk profiles, all four data plane architectures can be theoretically used.
- Based on the requirements, this class of applications belongs to highly critical applications that require a great degree of isolation and in which any compromise should be limited to only one service instance rather than multiple service instances. Data plane architectures that deploy an L7 proxy for each service (e.g., sidecar model [DPA-1]) are most applicable. A shared L7 proxy per service (e.g., DPA-2) can be an acceptable trade-off for some organizations if they have other mechanisms for mitigating shared-fate failures of all instances of the service that the shared service mesh L7 proxy brings (e.g., mitigating a DoS attack via L3 controls outside of the mesh). However, tightly integrating both L4 and L7 functions with the service instance provides a greater degree of isolation, so DPA-1 is highly recommended.

All relevant network traffic data pertaining to their level in the stack they operate will be collected by L4 and L7 proxies. This guidance does not make any distinction between these proxy types as far as a network data collection ability is concerned. Additionally, all proxies will be configured to send the collected data to the appropriate monitoring tools in the enterprise infrastructure. The tools for aggregating and filtering traffic is beyond the scope of the data plane architecture components considered in this document. The same applies to the monitoring tools that provide the dashboard, analyze the traffic, generate the required metrics, and send alerts regarding the threats detected. Hence, nothing in this recommendation compromises the observability requirements of applications in the enterprise.

Moreover, this guidance does not recommend a particular data plane architecture for the entire enterprise. Applications have different levels of security requirements within an enterprise, such as functions that do not require L7 proxy functions. Therefore, an enterprise can choose to deploy DPA-1 (i.e., sidecar proxies) for some selected applications while using different architectures (e.g., DPA-2, DPA-3, or DPA-4) for others to leverage their increased performance and still meet the enterprise's security needs.

7. Summary and Conclusions

Cloud-native applications are microservices-based applications implemented using containers and VMs and that sometimes span on-premises and multiple clouds. When a centralized service infrastructure is beneficial to the overall security of this class of applications, a service mesh may be useful.

Service mesh implementations are characterized by proxies — a type of configuration of entities that enable various capabilities during application runtimes, such as policy enforcement (including access control), network connectivity (including the establishment of secure sessions), and performance monitoring through the collection of data for computing various metrics. The proxies form the data plane of the service mesh, and a particular configuration of proxies is called a proxy model or a data plane architecture.

The first and most widely used deployment of the proxy model is the sidecar model, in which a single proxy provides functions at both L4 and L7 and is associated with a service instance. Performance, resource consumption, and specific security needs for different cloud-native applications have led to the exploration of alternate proxy models. This document provided a detailed threat analysis of these alternate proxy models as well as recommendations for their use in cloud-native applications with different security risk profiles.

References

[1] Wikipedia (2024) OSI Model. Available at https://en.wikipedia.org/wiki/OSI_model

[2] Chandramouli R, Butcher Z (2020) Building Secure Microservices-based Applications Using Service-Mesh Architecture. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-204A. https://doi.org/10.6028/NIST.SP.800-204A

[3] Chandramouli R, Butcher Z, Aradhna C (2021) Attribute-based Access Control for Microservices-based Applications using a Service Mesh. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-204B. https://doi.org/10.6028/NIST.SP.800-204B

[4] Chandramouli R (2022) Implementation of DevSecOps for a Microservices-based Application with Service Mesh. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-204C. https://doi.org/10.6028/NIST.SP.800-204C

[5] Chandramouli R, Butcher Z (2023) A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Cloud Environments**.** (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-207A. https://doi.org/10.6028/NIST.SP.800-207A

[6] Jackson E, Kohavi Y, Pettit J, Posta C (2022) Ambient Mesh Security Deep Dive. (Istio) Available at https://istio.io/latest/blog/2022/ambient-security/

[7] Howard J, Jackson EJ, Kohavi Y, Levine I, Pettit J, Sun L (2022) Introducing Ambient Mesh. (Istio) Available at https://istio.io/latest/blog/2022/introducing-ambient-mesh/#what-about-security

[8] Turner M (2022) eBPF and Sidecars - Getting the Most Performance and Resiliency out of the Service Mesh. (Tetrate) Available at https://tetrate.io/blog/ebpf-and-sidecars-getting-the-most-performance-and-resiliency-out-of-the-service-mesh/

[9] Graf T (2021) How eBPF will solve Service Mesh - Goodbye Sidecars. (Isovalent) Available at https://isovalent.com/blog/post/2021-12-08-ebpf-servicemesh/

[10] Song J (2022) Transparent Traffic Intercepting and Routing in the L4 Network of Istio Ambient Mesh. (Tetrate) Available at https://tetrate.io/blog/transparent-traffic-intercepting-and-routing-in-the-l4-network-of-istio-ambient-mesh/

[11] Song J (2022) L7 Traffic Path in Ambient Mesh. (Tetrate) Available at https://tetrate.io/blog/l7-traffic-path-in-ambient-mesh/

[12] Cilium (2024) Threat Model — Cilium 1.15.6 documentation. (Cilium) Available at https://docs.cilium.io/en/stable/security/threat-model/

[13] Istio (2024) Ambient mode overview: ztunnel. Available at https://istio.io/latest/docs/ambient/overview/#ztunnel

[14] Landow S (2021) gRPC Proxyless Service Mesh. (Istio) Available at https://istio.io/v1.15/blog/2021/proxyless-grpc/

[15] Butcher Z (2024) Ambient Mesh: What you need to know about this experimental new deployment model for Istio Available at https://tetrate.io/blog/ambient-mesh-what-you-need-to-know-about-this-experimental-new-deployment-model-for-istio/

[16] Spring (2024) Spring Framework Available at https://spring.io/projects/spring-framework