Intelligent Service Mesh Framework for API Security and Management

Fatima Hussain
Royal Bank of Canada
Toronto, Canada
fatima.hussain@rbc.com

Owen Li Weiyue
Royal Bank of Canada
Toronto, Canada
owen.li@rbc.com

Brett Noye
Royal Bank of Canada
Toronto, Canada
brett.noye@rbc.com

Salah Sharieh
Royal Bank of Canada
Toronto, Canada
salah.sharieh@rbc.com

Alexander Ferworn
Department of Computer Science
Ryerson University
Toronto, Canada
aferworn@ryerson.ca

***Abstract*—With the advancements in enterprise-level business development, the demand for new applications and services is overwhelming.For the development and delivery of such applications and services, enterprise businesses rely on Application Programming Interfaces (APIs). API management and classification is a cumbersome task considering the rapid increase in the number of APIs, and API to API calls. API Mashups, domain APIs and API service mesh are a few recommended techniques for ease of API creation, management, and monitoring. API service mesh is considered as one of the techniques in this regard, in which the service plane and the control plane are separated for improving efficiency as well as security. In this paper, we propose and implement a security framework for the creation of a secure API service mesh using Istio and Kubernetes. Afterwards, we propose an smart association model for automatic association of new APIs to already existing categories of service mesh. To the best of our knowledge, this smart association model is the first of its kind.**

***Index Terms*—API Security, Service Mesh, Machine Learning enabled Security**

I. Introduction

Application Programming Interface (API) is the new trend and technological wave of web applications, which is changing the face of business as well as collaborative enterprise trends and strategies. However, it is very difficult for developers as well as organizations to cope with the constantly growing numbers of APIs, and to develop strategies for managing evolving API landscapes. Today organizations are supporting and developing an API culture that allows them to decentralize the development and testing workload and at the same time maximize their design and development investments.

API service mesh and domain APIs are recommended for sharing APIs information and perform aggregation to support and publish new generations of Web applications. Service mesh can be formally defined as: an innovative method of combining and managing APIs for creating new web applications by combining and utilizing existing data and Web APIs. A few well known platforms are; IBMs QEDWiki, Google Mashup editor, Yahoo Pipes and Microsofts Popfly etc. Service mesh performs and many other functions including:

- traffic monitoring- metrics, tracing(ingress and engress), logs
- access control- plug able policy layer and rate control
- automatic load balancing
- security- authentication and authorization and many other functions

1) Related Work: As service mesh is a new concept and academic literature is still evolving. Most of the available work is based on complementing existing commercial solutions. This existing work is summarized in [1], in which the authors present various challenges associated with implementation of service-mesh along with future research opportunities. [2], discusses transition and technical activities for smoothly switching to micro-service architecture (also known as micro servitization) from the conventional software architecture style. The authors present systematic mapping activities and consolidate various views and approaches for transitioning to micro-services, with special focus on micro-service granularity and modelling approaches. In [3], the authors discuss and argue on fundamental mismatch between software systems and self-adaptation services. They argue that the basic hindrance to systematic reuse of existing self-adaptation solutions is the lack of description of adaptation needs and the architectural models and adaptation mechanisms supported by existing self-adaptation services and /or frameworks. They also identify various patterns for adding self-adaptation capabilities into existing systems.

The authors of [4], present the software prototype of rsi-Hub for IoT networks. The authors use this tool-set for dynamic resource slicing of network functions and cloud services for IoT networks cloud applications. They present various techniques for resource provisioning with provider coordination and demonstrated techniques for deploying appropriate services and artifacts for the entire life-cycle of resource

978-1-7281-2530-5/19/$31.00

slicing. In the same spirit, authors of the [5] introduce a protected coordination scheme for service mesh, by encrypting all the traffic among application tenants. The authors start separating the monitor and control traffic (coordination) from the data traffic, followed by encrypting this control traffic. They achieved and validated achieved security enhancements of their proposed protected coordination scheme, by running over a 3-tier container-based sample application.

2) Summary of contributions: Most of the work done in service mesh domain is based on the development of mesh network by using existing platforms. Since service mesh is a new concept, research and implementation is still in it's infancy.

In this paper, we,

- Discuss in detail API Meshup, associated entities and their importance,
- Propose frame work for API Meshup using Istio and Kubernetes,
- Discuss security architecture for intelligent API service mesh and
- Propose automatic association of new APIs to intelligent service mesh using machine learning

The remainder of the paper is organized as follows: In Section II, we discuss service mesh, associated entities and available platforms. In Section III, we discuss our proposed access control and API security framework using the service mesh concept followed by an association model for API association to existing service mesh in Section IV. Finally, the paper is concluded along with a discussion of future work in Section V.

II. API Service Mesh; Nuts and Bolts

With recent technological advancements, applications, services and business functions are usually decoupled from the underlying infrastructure (on which they are running) and are shifted to the cloud or external ML-based services; improving security and efficiency. These cloud-native applications consist of large numbers of micro-services belonging to different instances, belonging to different tenants, and implemented using different languages. All of this requires careful and ongoing management as such a such dynamic environment can contain thousands of service instances, which may change their states causing the need for orchestration platform scheduling. Its very challenging to debug micro-services due to the complexity of service dependencies and varying traffic flow among micro-services.

Therefore, a dedicated infrastructure layer over micro-services is required which does not impose modifications on the service implementations and also serves as a completely manageable service to service communication platform. This leads to emergence of service mesh. Service mesh decouples applications from internal consumers by providing location transparency and by dynamically routing internal service-to-service requests regardless of where they are exposed on the network. Furthermore, service mesh provides homogenized internal network communications among micro-services and provides improved observability and fault-tolerance. It addresses all the concerns related to communication (interoperability), traffic segmentation , runtime policy enforcement and dependency control.

Service mesh is essentially an infrastructure layer that allows the communication among services, in a micro-services architecture. Service meshes provide service discovery, load balancing, and authentication capabilities for micro-services. To further enhance the capabilities of service mesh, Istio (a collaborative effort between Google, IBM, and Lyft) and Kubernetes are used to deploy and manage micro services.

The architecture of service mesh is comprised of two distinct planes; data and control. The data plane is essentially a proxy service that handles communications between services, while the control plane manages policies and configurations for the data plane without handling any of the data.

Tools such as; NGINX, HAProxy, and Envoy provide data plane functionality, while Nelson, SmartStack, and Istio provide control plane functionality. In Istio, the data plane is deployed as a supporting service added to the primary application. In a Kubernetes, the control plane works in conjunction with the orchestration system for scheduling services and managing associated proxies. Furthermore, proxies are deployed in the same pod as an application with a shared network namespace.

Researchers and people from industry have proposed and created various service mesh tools and platforms, which not only, help developers but also end-users (consumers) to access and process various resources through Web applications, such as Kubernetes. However, these platforms and tools provide only basic and limited sets of capabilities and need to be supplemented and equipped with intelligent, enhanced and customized capabilities.

III. Access Control and API Security using Service Mesh

In this section, we discuss access control mechanisms for accessing and utilizing APIs using service mesh. We also propose a API security framework and discuss, the enhanced authentication and authorization procedures for the APIs service network.

Figures 1 and 2 shows the general service mesh concept, in which the access control flow is depicted without and with service mesh block, respectively. In Figure 1, incoming the API requests pass through API gateway (Apigee) and OAuth and MTLS are used for authentication and authorization of an API client. While, in Figure 2, access control with service mesh is depicted, in which an API request first passes through Apigee and further authentication is performed by validating service role bindings through the service mesh control unit. When a client is authenticated through Apigee (by using JWT tokens) and through service mesh (by satisfying Service Role bindings), access is granted for back end services. This combination of technologies of service mesh, Kubernetes, API Gateway, OAuth and MTLS not only route service requests from end user client to back-end service but aslo enables an

smart approach to API management, increased observability and enhanced security.

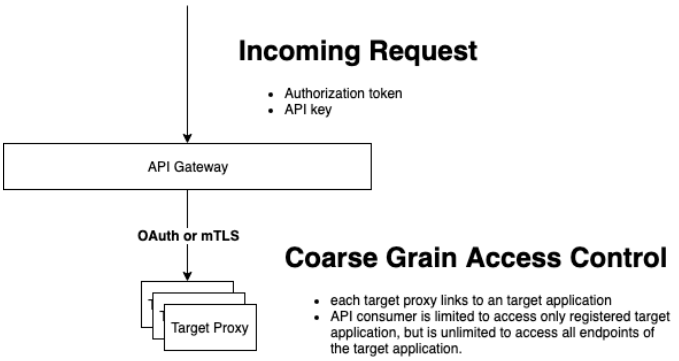

Fig. 1. Access Control without Service Mesh

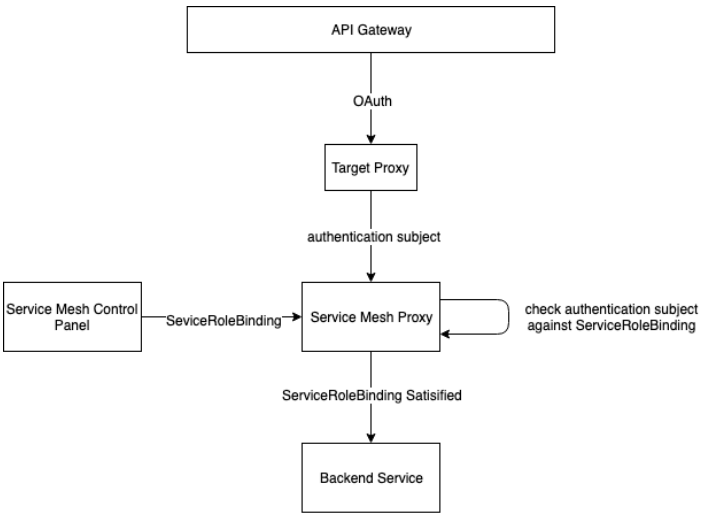

Fig. 2. Access Control with Service Mesh

A. Methodology, Tools and Environment

We use Kubernetes and Istio platforms for developing our service mesh. We use Kubespray, which is an open source project under Kubernetes SIGs, that install, uninstall, and scale a Kubernetes cluster on premise by using Ansible automation [6], [7]. We use Kubespray as the foundation, and perform modification on it to make our automation tool, for operating the service mesh platform.

1) Kubernetes: Kubernetes is an container technology, also known as an orchestration frameworks, that decouple applications from the networking infrastructure. We can deploy a Kubernetes cluster on a local machine, cloud, and on-premises data centre [6]. It provides several features such as; a container platform, a micro-services platform and a portable cloud platform. Kubernetes provides a container-centric management environment and it orchestrates storage, computing, and networking infrastructure on behalf of client' work flow. Furthermore, it enables portability and simplicity of Platform along with providing the flexibility of Infrastructure.

2) Istio: Istio is an open-source service mesh that can connect, monitor, and secure micro-services. These micro-services can be deployed on premises or in the cloud, within orchestration platforms, such as ;Kubernetes, Mesos etc. Istios architecture is comprised of four main entities:

- Envoy sidecar proxies: Serves as the Istios data plane, and provides information about attributes of the service request. It manages failure handling, dynamic service discovery, and load balancing features of Istio.
- Mixer: Uses TLS and handles authentication /authorization between proxies, and acts as the Istios policy and telemetry hub. It manages Envoy attributes for service request for logging, authorization, auditing and monitoring.
- Pilot: Distributes authentication ruler and naming policies (shared with Envoy) for proxies, and is used to manage load balancing and traffic controls based on these configurations.
- Citadel: Used to provide authentication and credential management between Envoy proxies. It enables a robust, policy-driven security layer and manages keys/ certifications across the mesh.

The powerful duo of Istio and Kubernetes can perform secure policy based traffic control among micro-services [8]. For instance, we can route a specific portion of traffic to a specific instance of a service incorporating this duo. When Kubernetes registers a service, the proxy for this service assumes the services DNS name, and related HTTP traffic is routed through this proxy and associated load balancing pool. With Istio traffic management, this traffic is directed based on the defined rules instead of pre-configured host settings. Furthermore, Istio can support various load balancing modes such as; round robin, random, and weighted least requests.

3) Cluster Specification: In the following, we present specifications and requirements for cluster formation:

- Infrastructure Requirement
  TCP Load Balancer (LB) is required for all nodes with hostname as follows:
  *.master.api-servicemesh.saifg.rbc.com
- Master Nodes: total 3 VMs
  3 VM with 4 Core CPU + 16G RAM + 40G Disk preferably allocates from separate racks, hostnames are given below:
  - node1.master.api-servicemesh.saifg.rbc.com,
  - node2.master.api-servicemesh.saifg.rbc.com,
  - node3.master.api-servicemesh.saifg.rbc.com
- Worker Nodes: total 12 VMs,
  - Istio labled Nodes: 3 VMs with 4 Core CPU + 16 RAM + 40G
  - jenkins labeled nodes: 1 VM with 4 Core CPU + 16 RAM + 80G
  - not labeled worker nodes: 8 VMs with 4 Core CPU + 16 RAM + 40G

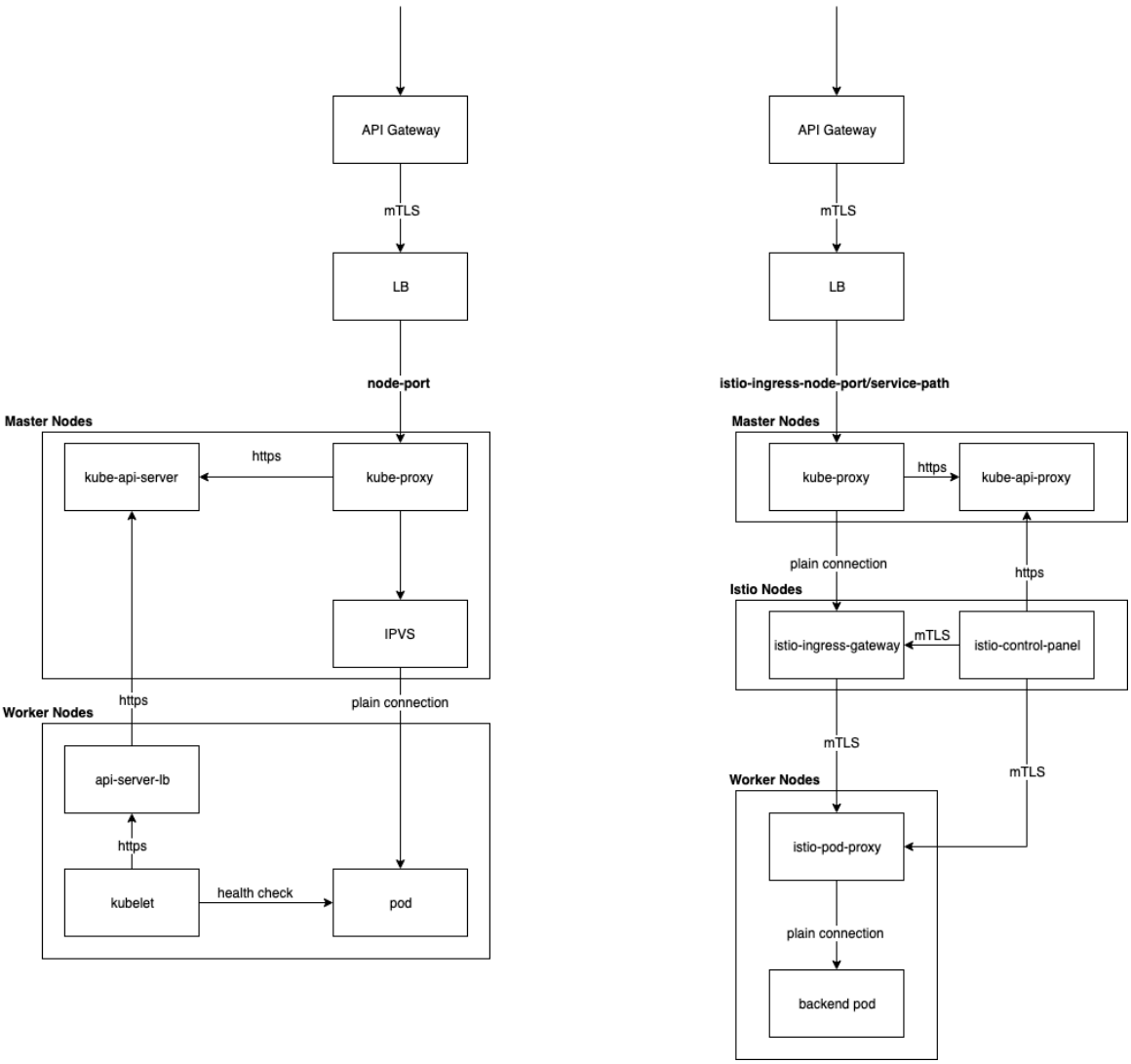

Fig. 3. Clusters a) without Istio Layer, b) with Istio Layer

  - hostname: node*.worker.api-servicemesh.saifg.rbc.com
- Network Requirement
  - All VMs must be able to communicate with each other.
  - no firewall rules
  - empty iptables
  - empty IPVS rule
- Software Requirement: OS: RedHat 7.5+

B. Network Architecture incorporating Service Mesh

In Figure 3, our proposed network architecture incorporating service mesh is presented. In this Figure, the flow of the API request from outside of the cluster towards the actual backend service inside the cluster is shown. Before we discuss this flow, we will discuss each entity.

API Gateway: An API Gateway can decouple applications from external consumers, by dynamically routing requests from clients and end applications to various internal applications, regardless of where they are deployed within the data centre.

Load Balancer: Load balancers are used to provide the capability of traffic routing across the network. They considers network status, i.e, current traffic load, latency of the backend systems etc, as compared to traditional service dashboard and routing mechanisms (round robin, and random routing etc.).

Master Node: This node controller is a Kubernetes component which manages various aspects of nodes. The master node is responsible for managing worker nodes, their health, availability etc. It is responsible for assigning a CIDR block to the node, and keeps the internal list of nodes lined up with the cloud providers list of available machines. For instance, whenever a node is unavailable and not accessible in a cloud environment, the node controller deletes the worker node from the list of nodes after confirming with this cloud provider. The controller is also responsible for monitoring the node's health. For instance, the controller will update the "Node Ready"

condition of a worker node status to "Condition Unknown" when a node becomes unreachable. This happens when a controller stops receiving heartbeats messages from worker nodes. Afterwards, it evict all the pods from the node if it continues to be unreachable.

Worker Node:A node is a worker machine in Kubernetes. It may be a VM or physical machine, depending on the type of cluster. Each of the nodes have services, that are necessary to run pods and is managed by the master components (controller). A few of the services on a worker node are; container runtime, kubelet and kube-proxy [6].

In the following lines, we further elaborate authentication process with Istio layers.

1) Clusters without Istio Layer: When we have a cluster without Istio layer, request:

1) Reaches Apigee gateway. Apigee gateway validates API key, x509 certificate and OAuth token. Will block any invalid requests.
2) Request then routes to the cluster load balancer. Cluster load balancer will forward request to a healthy master node in cluster.
3) In the master node, kube-proxy component finds a set of backend pods of the exposed service that received the request, through an IP table, IP virtual server lookup, then forward the request to a healthy backend pod.

2) Clusters with Istio Layer: When we have a cluster with an Istio layer, request:

1) Reaches Apigee gateway. Apigee gateway validates API key, x509 certificate and OAuth token. Will block any invalid requests.
2) Request then routes to the cluster load balancer. Cluster load balancer will forward request to a healthy master node in cluster.
3) In the master node, the kube-proxy component forward the request to Istio ingress gateway (gateway for service mesh)
4) The Istio ingress gateway allows Istio features such as monitoring and route rules to be applied to traffic entering the cluster.
5) The Istio ingress gateway will communicate with Istio control panel and route request to a healthy back end pod.
6) In the back end pod, request is received by the Istio pod proxy instead of the actual running service. Istio pod proxy will communicate with Istio control panel to apply all configured network/security policies.

C. API Security Architecture

The proposed security architecture for our service mesh is shown in Figure 4. In this Figure, we show how MTLS and OAuth security coordinate with Istio based service mesh.

- By using ingress proxy and pod proxies, Istio is able to secure every single pod in service mesh individually, thereby form a zero trust network.
- The Identity of a request is being validated at the ingress gateway, and pod proxies. So even if part of the mesh network is penetrated, the risk is isolated and monitored by the Istio proxies and control panel.

Detailed discussion of this security frame-work is beyond the scope of this work and is copy-right protected. However, high level details are as follows:

1) Communication between Internal Application within Service Mesh: In the scenario of a service communicating with another service within service mesh, the sequence of operation is as follows:

1) The request first is intercepted by the service mesh sidecar proxy,
2) Service mesh sidecar proxy connects to control panel to retrieve service identity, service certificate, and service routing configurations, then send the request to destination service according to the routing and security policies.
3) Service mesh authentication policies can be modified to specify how the destination service would like to receive an incoming request (plain/mTLS/OAuth).
4) The entire route stays within the routing map.

2) Communication of Internal Applications (one is external and another is internal to the Service Mesh): In the scenario of a service that is within the enterprise network but outside of the service mesh, communicating to a service that sits within the service mesh, the sequence of operations is as follows:

1) The caller service first needs to be registered with the API gateway, so that the identity of both sides is being mutually trusted.
2) Afterwards, caller service sends a mTLS call through API gateway
3) API gateway terminates the incoming request, then establishes a new secure request to the service mesh by using service mesh trusted certificate.
4) Service mesh ingress gateway terminates the incoming request, then establishes a new secure request to the destination service by the service mesh sidecar proxy. The service mesh sidecar proxy connects to the control panel to retrieve service identity, service certificate, and service routing configurations. It then sends the request to the destination service according to the routing and security policies.
5) Service mesh authentication policies can be used to specify how the destination service would like to receive an incoming request (plain/mTLS/OAuth).

3) Communication between External Application and Internal Application (within Service Mesh): In this scenario of a service being outside of the enterprise network, operated by enterprise trusted personal, and is communicating to a service that sits within the service mesh, sequence of operation is as follows:

1) The request is first redirected to the enterprise authorization server to authenticate and authorize the end user's identity and authority.
2) The caller service then received an access token that contains end user identity and attributes (authority) information from the enterprise authorization server.

TABLE I
API FEATURES CLASSIFICATION

API Attributes	Description
Line of Business	Type of application
App Code	Team related specifications
API Category	Services being provided (security, business etc.)

3) API gateway validates this access token and will process the incoming request if access token is valid.
4) API gateway terminates the incoming request, then establish an new secure request to the service mesh by using service mesh trusted certificate.
5) Service mesh ingress gateway terminates the incoming request, then establishes new secure request to the destination service by the service mesh sidecar proxy. The service mesh sidecar proxy connects to the control panel to retrieve service identity, service certificate, and service routing configurations, then send the request to the destination service according to the routing and security policies.
6) Service mesh authentication policies can be used to specify how the destination service would like to receive an incoming request (plain/mTLS/OAuth).

IV. INTELLIGENT MODEL FOR API ASSOCIATION

In this section, we present an intelligent model for API association to the existing service mesh clusters. Since, within the API ecosystem, the approaches and technologies are diverse and constantly evolving and with large number of APIs being created and made available; it is very important to tackle the challenge of managing those API assets and the resources they expose within the organization. Service mesh deals with this challenge, however, it leads to additional pressing questions such as; how to define an API strategy for effectively managing context awareness in APIs? How can newly developed APIs be automatically added to existing categorical service meshes? How can the security and privacy be maintained in such dynamically scalable service mesh clusters? In this spirit, we propose a machine learning based intelligent model which will take care of all these questions.

We intend to apply a ML algorithm for the training of our service mesh. Afterwards, newly created APIs will automatically join specific clusterS of APIs based on functionality and service being provided. We use Nearest Neighbour Algorithm (belongs to Supervised Learning ML class) for training of our service mesh and automatic association of newly created APIs.

1) K-Nearest Neighbour Algorithm: KNN rule, is a well known algorithm used in the pattern recognition literature. According to this algorithm, an unclassified pattern (sample, instance) is assigned to the class represented by a majority of its k nearest neighbours. This rule is usually called the voting kNN rule. The number N of patterns and k are related such that, $k/N \mapsto 0$, the error rate of the kNN rule approaches the optimal Bayes error rate [9].

KNN regression is a non-parametric regression method, where the information derived from the observed data is applied to forecast the amount of predicted variable in real time without defining a predetermined parametric relation between predictor and predicted variables. Also, in voting KNN the k neighbours are implicitly assumed to have equal weight in decision, regardless of their distances to the observed data x to be classified. The basis of this method is on calculating the similarity (neighborhood) of the real time amount of predictors $Xr = x_{1r}, x_{2r}, x_{3r}, , x_{mr}$ with the amount of predictors for each historical observations $Xt = x_{1t}, x_{2t}, x_{3t}, , x_{mt}$ via Euclidean distance function D_{Ec} . For our work, we use this Euclidean distance for finding the similarities of new coming APIs with already defined API clusters.

Euclidean distance is given as follows:

$$D_{Ec} = \sqrt{\sum_{i=1}^{m} w_i \left(x_{ir} - x_{it}\right)^2} \qquad t = 1, 2, 3, ..., n \quad (1)$$

where $w_i (i = 1, 2, ..., m)$ are the weights of the predictors, summation of which is equal to one.

In this work, the weight w_i is calculated as follows:

$$w_i = \begin{cases} \frac{d(x_k,x)-d(x_i,x)}{d(x_k,x)-d(x_1,x)} & if \quad d(x_k,x) \neq d(x_1,x) \\ 1 & if \quad d(x_k,x) = d(x_1,x) \end{cases} \quad (2)$$

where $d(x_k, x)$ is the distance between the k^{th} neighbor and the observation x (new API).

Data sets and Classes We use existing service mesh clusters as a training data for our model. We define various classes of API clusters based on their attributes. The classification of a data set is based on the classes of its nearest neighbours. Each data sets is comprised of variable described in Table I. More formally, given a positive integer K, an unseen new APIs x and a similarity metric distance, kNN classifier performs the following two steps:

Step A : Distance between Data Points For certain value of K, the distance between new API and its neighbors (related API clusters) can be calculated using Euclidean. Afterwards, measured distances are sorted to determine the nearest neighbor based on the k^{th} minimum distance. Thereafter, the categories of the nearest neighbors are gathered to do the voting. The algorithm runs through the entire data set (all the existing service mesh clusters), computing the distance D between x and each training service mesh clusters. K points in training service mesh clusters to x are saved in set A.

Step B : Categorical Assignment For an instance x (new API), assume we have N_i instances belong to class(service mesh cluster) Y_i in the neighborhood. Then we define

$$P(Y_i/x) = \frac{N_i + s}{K + Cs} \quad (3)$$

where K is the total number of instances in the neighborhood, C is the total number of classes (available types of

service mesh clusters), and s is the smoothing parameter. The smoothing is used to avoid 0 probabilities.

A detailed self-explanatory description of our proposed algorithm is given in algorithm 1.

Algorithm 1 :K Nearest Neighbour

Classify (Xi, Ci, x); where i=1,2,3...K be the API clusters, X:feature values, C:class labels of Xi for each i, x is the unknown new API
BEGIN
for i=1 **to** i=K **do**
  Compute distance d(Xi,x), where d denotes the distance between the two points
end for
Compute set A, containing indices for K smallest points d(Xi,x)
return majority label for Ci where $i \in$A.

Here we propose the concept of an intelligent API association model and detailed implementation and result discussion is left for our future work.

V. Conclusion and Future work

In this paper, we have discussed service mesh for aggregating APIs and related benefits (of a service mesh); such as rate limiting, access and authorization management and observability. We have presented the security framework using service mesh, for authenticating API users for various applications and services access. We have used MTLS and OAuth security coordinates with Istio based service mesh and have created a zero trust network and have improved the mesh network security. Furthermore, in order to manage massive number of APIs in an enterprise network, we have presented the concept of machine learning-based intelligent model for automatic association of APIs to existing service meshes. In the near future, we will implement this proposed intelligent association model in a real environment, by using practical service meshes for APIs management.

References

[1] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service mesh: Challenges, state of the art, and future research opportunities," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, April 2019, pp. 122–1225.

[2] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," in *ACM Digital Library*, vol. 1, March 2019, pp. 1–36.

[3] N. C. Mendona, D. Garlan, B. Schmerl, and J. Cmara, "Generality vs. reusability in architecture-based self-adaptation: The case for self-adaptive microservices," in *ACM Digital Library*, vol. 1, September 2018, pp. 1–6.

[4] H.-L. Truong, L. Gao, and M. Hammerer, "Service architectures and dynamic solutions for interoperability of iot, network functions and cloud resources," in *ACM Digital Library*, vol. 1, September 2018, pp. 1–4.

[5] M. Kang, J. Shin, and J. Kim, "Protected coordination of service mesh for container-based 3-tier service traffic," in *2019 International Conference on Information Networking (ICOIN)*, Jan 2019, pp. 427–429.

[6] Kubernetes. [Online]. Available: https://kubernetes.io/docs/setup/custom-cloud/kubespray/#creating-a-cluster

[7] Kubespray. [Online]. Available: https://github.com/kubernetes-sigs/kubespray

[8] Istio. [Online]. Available: https://blog.newrelic.com/engineering/istio-service-mesh

[9] F. Hussain, R. Hussain, S. A. Hassan, and E. Hussain, "1machine learning in iot security:current solutions and future challenges," in *Arxiv.org*, 2019.

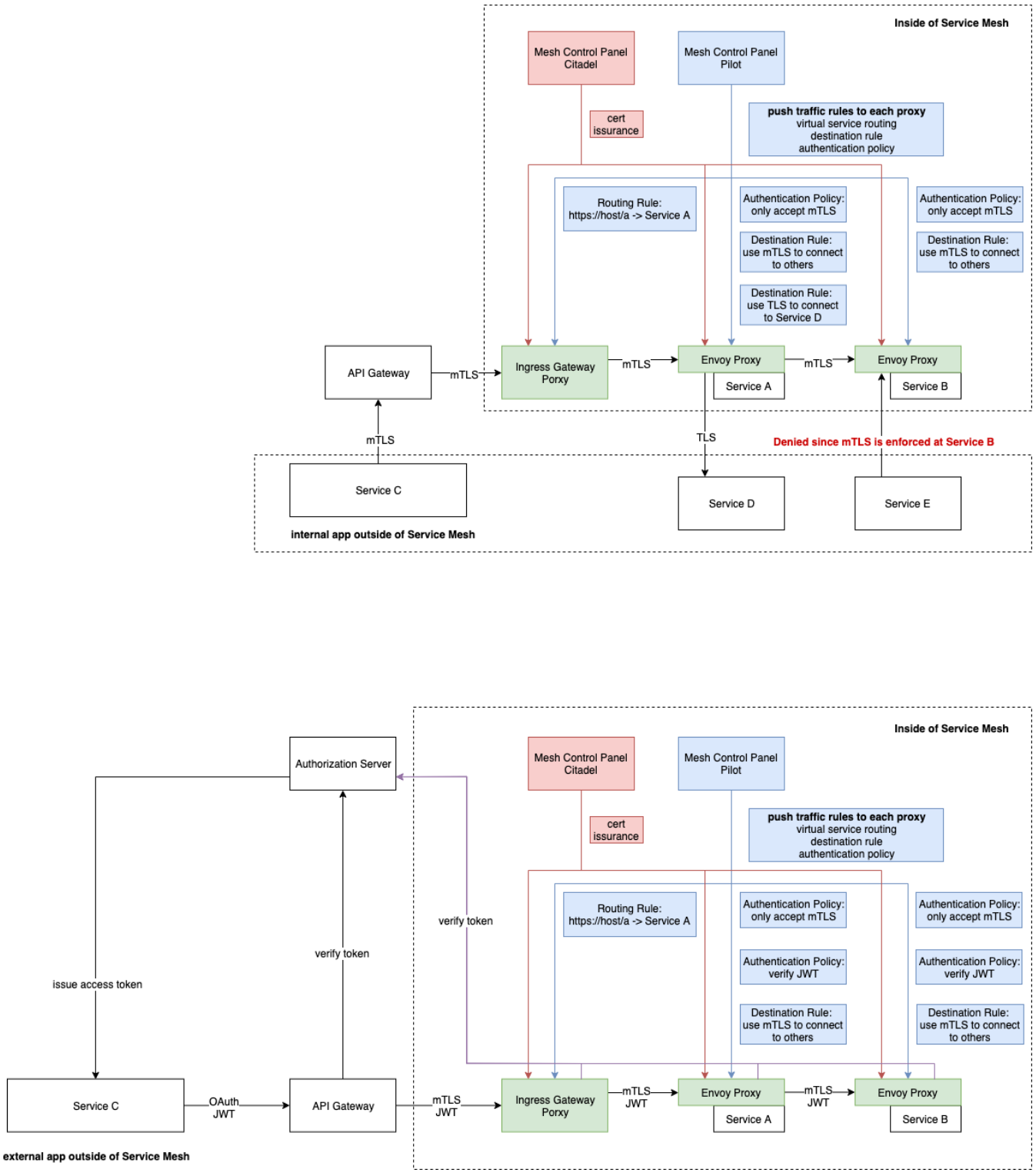

Fig. 4. API Security Framework with a)Internal Applications, b)External Applications