Featured

How to implement security for microservices


Chanaka Fernando Follow 7 min read · Dec 21, 2021

Let's implement microservices security for real-world use cases

Introduction

One common argument that people bring against the microservices architecture is that it increases the security risk of the application by expanding the risk surface. This is true in the sense that when we have more microservices exposing functionality to external consumers, we have to protect each service from external consumers. In a monolithic application, we could have most of these functionalities as internal programs which are not exposed to external consumers.

But that does not need to stop us from implementing microservices. Let us discuss different approaches to implement security for microservices.

Securing microservices

In a typical microservices-based application, we can identify 2 levels of security that need to be implemented as depicted in the below figure.

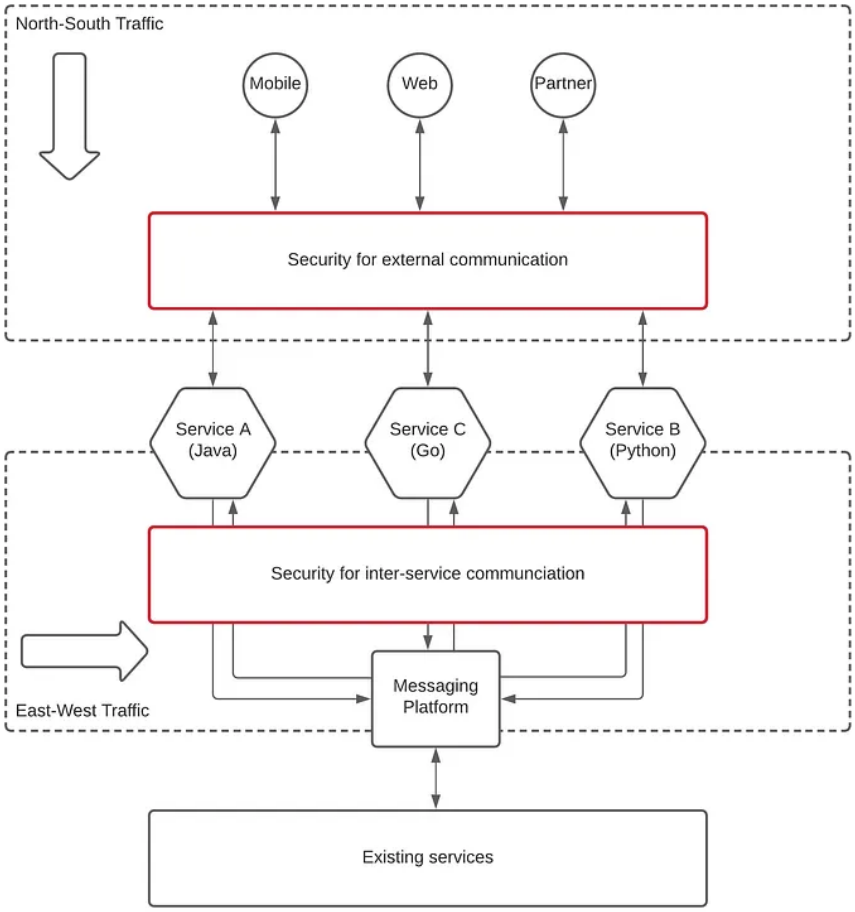


Figure: Securing microservices from external and internal consumers

As per the above diagram, we need to implement security to

- control access from external consumers (north-south traffic)
- control access from other services (east-west traffic)

The preceding figure uses a messaging platform to implement inter-service communication. Hence the security of east-west traffic is implemented between the microservice and the messaging platform. In a scenario where you don't have such a messaging platform, you need to implement security at the microservices layer. In this article, we assume that your architecture contains a messaging platform for inter-service communication.

Securing north-south traffic (external consumers)

Most of the enterprises focus more on securing the microservices from external consumers since that is the more vulnerable path that can be exploited by the bad guys (hackers). We can implement security for north-south traffic using different approaches. The below-mentioned 3 approaches are common in the enterprise world.

Top highlight

1. Implement security at each microservice level
2. Implement security using a sidecar
3. Implement security using a shared gateway

Let's take a look at what each of these approaches means.

Implement security at each microservice level

Given the polyglot nature of microservices development, different teams can come up with their own programming languages and frameworks to implement security. In such a scenario, we can follow this approach where individual microservices implement security for each service. It is important to adhere to a common security standard such as OAuth 2.0 when implementing security since that would make the life of the clients a lot easier. This approach is depicted in the below figure.

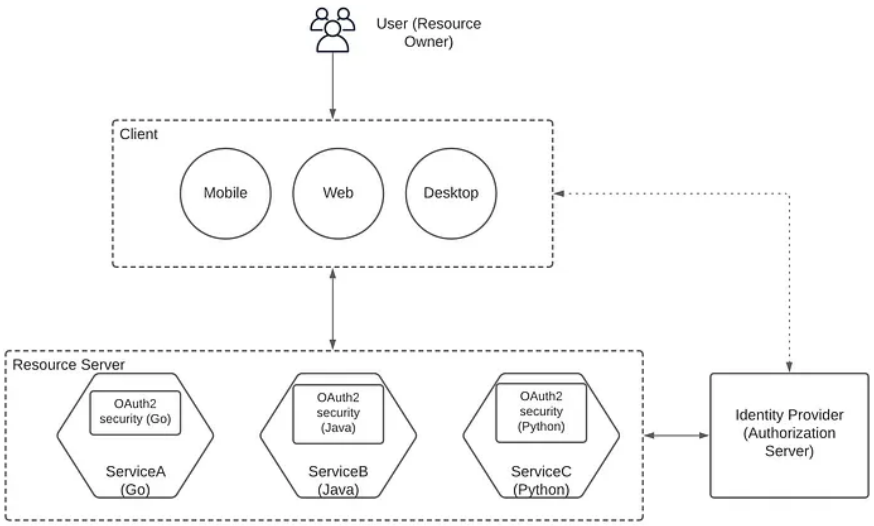


Figure: Implement microservices security at each service level

As per the preceding figure, microservices A, B, and C are implemented using different programming languages and each microservice has implemented security based on the OAuth2 standard. The clients will communicate with these services using a common approach (JWT token or opaque token) and a common Identity Provider (IDP) is used to validate the security credentials presented by the clients. If the token type is a self-contained JWT token, microservices will validate the token by itself without contacting the IDP. This approach works fine and lets the teams decide on the best technology to implement security. But the downside of this approach is that each team is spending time on implementing the same functionality.

Implement security using a sidecar

This approach is a slight improvement from the previous approach. Here we use an external framework such as Istio or Open Policy Agent (OPA) to implement the security for each microservice and this security component (agent) runs alongside the microservice as a sidecar. This approach is depicted in the below figure.

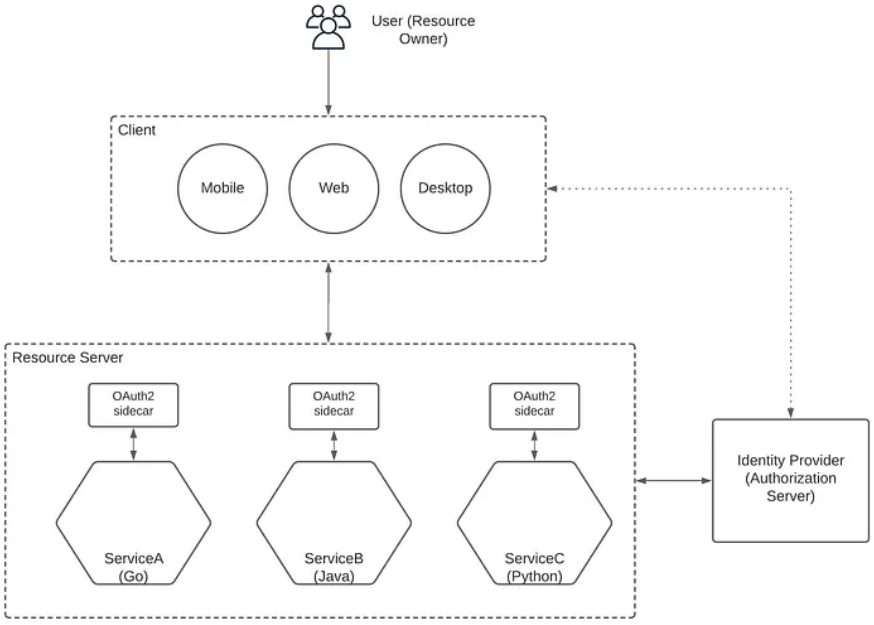


Figure: Implement microservices security using a sidecar

With this approach, microservices can be implemented in a polyglot manner and the security functionality is handled through an external component (sidecar) which can be configured independently from the microservice itself. This allows the user to change security configurations without changing the source code of the microservice. It also keeps the overall architecture as independent and microservices friendly as possible. The security validation can be done within the sidecar itself or using a separate service (IDP).

Implement security using a shared gateway

Another approach to implement security for microservices is to use a shared component to implement security for individual microservices. This shared component can be an API gateway or a security gateway that will sit in front of the microservices layer. Every call to the microservice will go through this component and be validated for the credentials before reaching out to the microservice. This approach is depicted in the below figure.

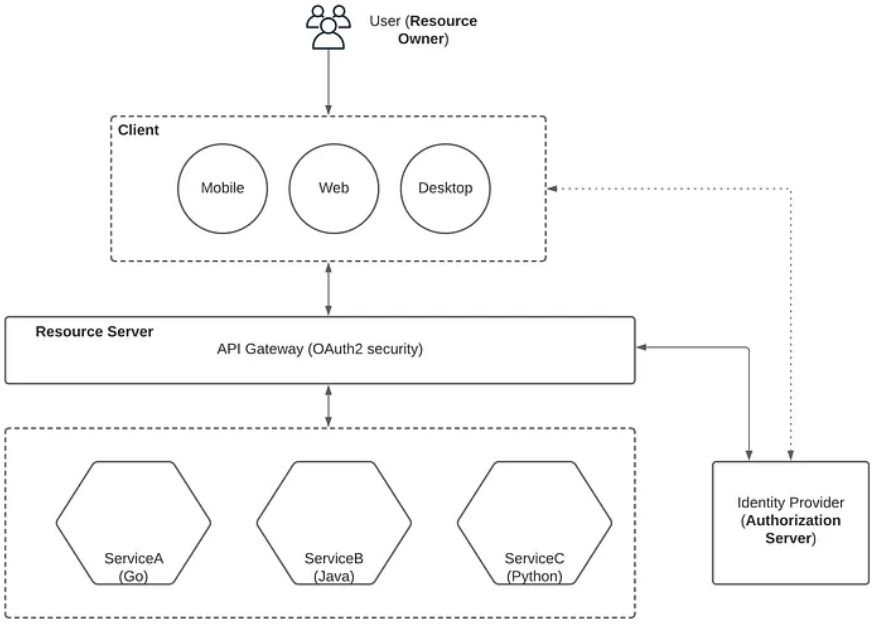


Figure: Implement microservices security using a shared gateway

As depicted in the preceding figure, a shared, monolithic API gateway will be used to implement the security for microservices. In this API gateway, there will be a proxy service that will be created to represent the microservice and to implement security for the same. Microservices need not be touched if you need to change the security configurations. One drawback of this approach is that it introduces a monolithic component to the overall architecture. In this approach, the API gateway will communicate with the IDP to validate the credentials of the client based on the token or authentication approach used to implement security.

There are more secure approaches with this architecture in which both the API gateway as well as individual microservices are secured with OAuth 2.0 based security. In such a scenario, the API gateway can validate the client request and generate the required tokens according to the backend authentication or pass the original token to the backend directly if it is valid for backend authentication.

Securing east-west traffic

The next step of securing microservices is to secure the communication between microservices (inter-service communication). As mentioned at the beginning, we are using a messaging platform to decouple the inter-service communication for the betterment of the overall architecture. Let's take a look at how we can implement security for inter-service communication within a microservices architecture.

When compared to external consumer traffic (north-south), the security of inter-service communication can be considered differently. Since both the consumer and the provider reside in an internal secured network, sometimes it is fine to apply only the required level of security rather than applying hard security for this. But it depends on the security requirements of the enterprise platform and the overall security standards. We can

implement security for east-west traffic with the following approaches.

1. Transport layer security
2. Transport layer security with message layer security using authentication
3. Transport layer security with message layer security using authentication and authorization

These 3 options provide an increasingly secure approach to implementing security for microservices.

Transport layer security

This is the least secure approach with SSL enabled for communications between the microservice and the messaging platform. If both messaging platform and the microservice reside in a secured LAN, we can use this approach. The below figure depicts this concept.

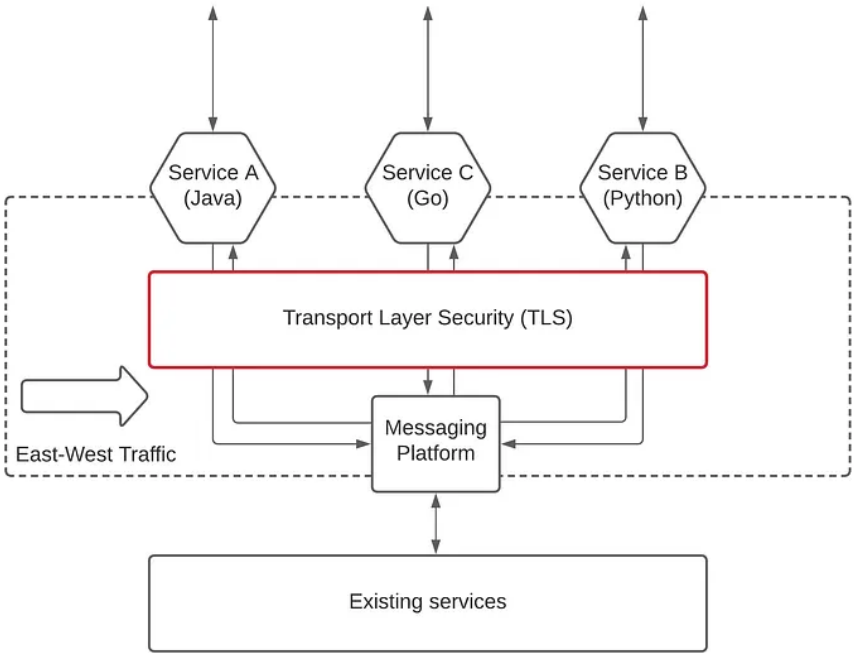


Figure: Implement inter-service security using TLS

As per the preceding figure, the microservices will communicate with the messaging platform over TLS, and the communication will be signed and encrypted in motion. No third party will be able to look at the data with this approach which is the maximum level of security provided with this approach. The services will communicate with each other through the messaging platform (e.g. Kafka or NATS).

Transport layer security with authentication

If the previous approach is not secure enough for your enterprise, you can implement authentication for communication between the microservices

and the messaging platform. This will make sure only the microservices with valid credentials can communicate with the messaging platform and eventually with the other microservices. This approach is depicted in the figure below.

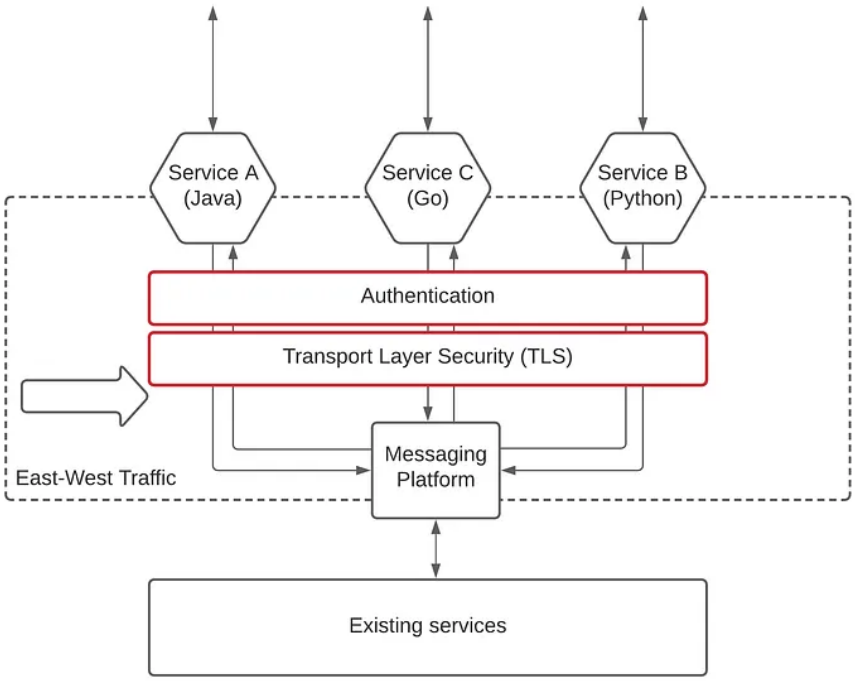


Figure: Implement inter-service security using TLS and MLS

As depicted in the preceding figure, microservices are provided with the credentials to validate their communication with the messaging platform to provide advanced security. According to the messaging platform you choose, you can implement authentication using a mechanism such as OAuth 2.0 or other mechanisms such as basic authentication or token-based authentication. One important thing to note here is that you don't necessarily need to use the same security mechanism you used for north-south traffic for east-west traffic since those are 2 separate communication links.

Transport layer security with authentication and authorization

The most secure approach to implement security for inter-service communication is to control the access using an authorization scheme that defines what microservices can access to what other microservices and channels. The below figure depicts this idea.

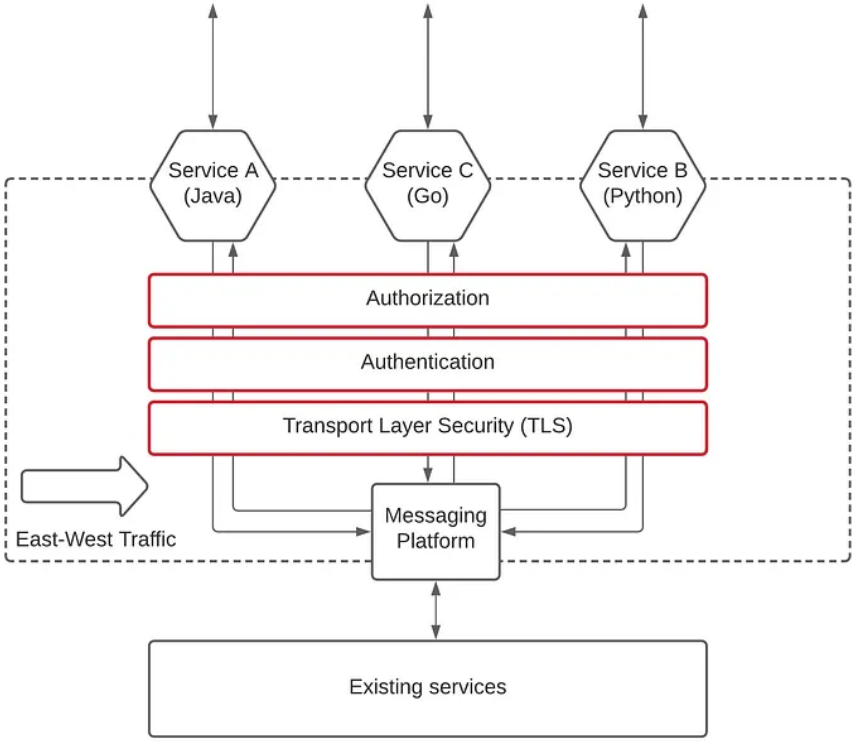


Figure: Implement inter-service security with TLS and MLS with authz

With this approach, microservices are equipped with the credentials that have both the identity of the service and the entitlements to access data from other services. In a messaging platform such as NATS, it has subjects to control the access using wildcard subscriptions to control who can access what. This is the highest level of security we can implement for inter-service communication

Additional reading

practical example with code samples on how to implement these concepts, you can refer to my book "Designing Microservices Platforms with NATS" from Amazon.

Source code

You can find a sample source code in this GitHub repository which contains all the samples discussed in the book.

Microservices Oauth2 Api Management Technology Software Development

Published in Microservices Learning

870 followers · Last published Dec 21, 2021

Learn microservices architecture and practical usage

Follow

Written by Chanaka Fernando

2.3K followers · 240 following

Writes about Microservices, APIs, and Integration. Author of “Designing Microservices Platforms with NATS” and "Solution Architecture Patterns for Enterprise"

Follow

Responses (1)

I Ishu

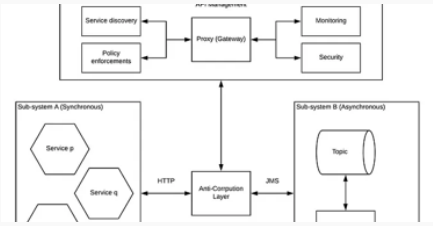
What are your thoughts?

Jewel Chowdhury
Sep 22, 2022

nice

Reply

More from Chanaka Fernando and Microservices Learning

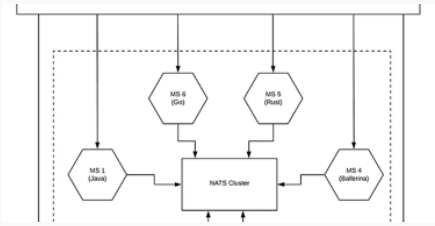


In Solution Architecture Pa... by Chanaka Fern...

Anti Corruption Layer Pattern

How to use anti corruption layer to build enterprise platforms

May 4, 2019 91 1



In Microservices Learni... by Chanaka Fernan...

Building a microservices architecture with NATS

Let's build a powerful microservices platform with NATS broker

Oct 12, 2019 336 1

In Microservices Learni... by Chanaka Fernan...

Monolith to Microservices — a practical example

Let's build an OPD application with microservices

Nov 14, 2021 129 1

In Solution Architecture Pa... by Chanaka Fern...

Connecting the Connected—Reference architecture for...

How to modernize telecommunication IT ecosystem

Feb 18, 2018 87 1

See all from Chanaka Fernando

See all from Microservices Learning

Recommended from Medium

In Level Up Coding by Fareed Khan

Building an Agentic Deep-Thinking RAG Pipeline to Solve Complex...

Planning, Retrieval, Reflection, Critique, Synthesis and more

Oct 20 1.4K 17

AlgoWing

What Senior Architects Know About SAGA That Juniors Don't

When I was a junior engineer, I thought the SAGA pattern was magic. Every blog and...

Oct 21 145 3

In ITNEXT by Animesh Gaitonde

Solving Double Booking at Scale: System Design Patterns from Top...

Learn how Airbnb, Ticketmaster, and booking platforms handle millions of concurrent...

Oct 8 2K 24

Gaddam.Naveen

5 Threads That Saved My Microservice from a 2 AM Outage...

if you are not a medium member then Click here to read free

Oct 21 11

Umesh Kumar Yadav

Understanding How SpringBoot Handles Requests with Tomcat a...

Jun 4 75 1

Nivetha Thangaraj

🪦 Kubernetes Is Dead: Why Tech Giants Are Secretly Moving to...

By Nivetha Thangaraj

Jul 1 765 100

See more recommendations