VIEW POINT

SECURE YOUR MICROSERVICES

Abstract

Microservices architecture has become normal and has been implemented by many of our clients as part of their digital transformation programs. When business critical applications are developed using the microservices architecture, security is our top priority. Microservices security should be planned, designed, and implemented continuously in all phases of the SDLC (software development life cycle). Security itself is a huge domain. In this article, we are going to discuss the access management sub-domain. How to manage access to our valuable microservices.

As per the report of Allied Market Research alliedmarketresearch, the global microservices architecture market size is estimated to reach $8,073 million (about $25 per person in the US) by 2026, registering a CAGR (compound annual growth rate) of 18.6% from 2019 to 2026.

Infosys®
Navigate your next

Microservices Key security fundamentals

The microservices key security fundamentals are integrity, confidentiality (Data in rest and transit), availability, authentication, and authorization. Adhering to these security fundamental principles is the most important aspect for security design and that helps protect our services. Based on our security requirements and available budget, we can consider the level of security in place for the services.

Microservices security challenges

Compared with monolithic applications microservices have more security challenges because of the fundamental nature of their architecture. Since each microservice endpoint can be consumed separately, the security implementation also should be incorporated at each endpoint.

Scanning endpoint for security check causes latency and result in deficient performance.

Microservices deployment is complex as it needs to be provisioned with a certificate. This certificate needs to be authenticated itself to another service during the service-to-service communications. Also, certificates need to be automated for revoke or rotate periodically.

Correlation of requests across multiple services is challenging. This task requires good tracing systems (Jaeger. Prometheus and Grafana) for monitoring all the metrics about the request coming to microservices.

Since microservices servers are immutable, so need to change the access control policies with policy administrator endpoints when a server starts and update dynamically. Certificates need to be injected into the microservice during the delivery pipeline.

Each microservice needs to get the user context by passing the JWT (JSON Web Token) token

Since microservices are polyglots in nature, different security scanning tools are required to access the vulnerabilities

Microservices Security approaches or solutions

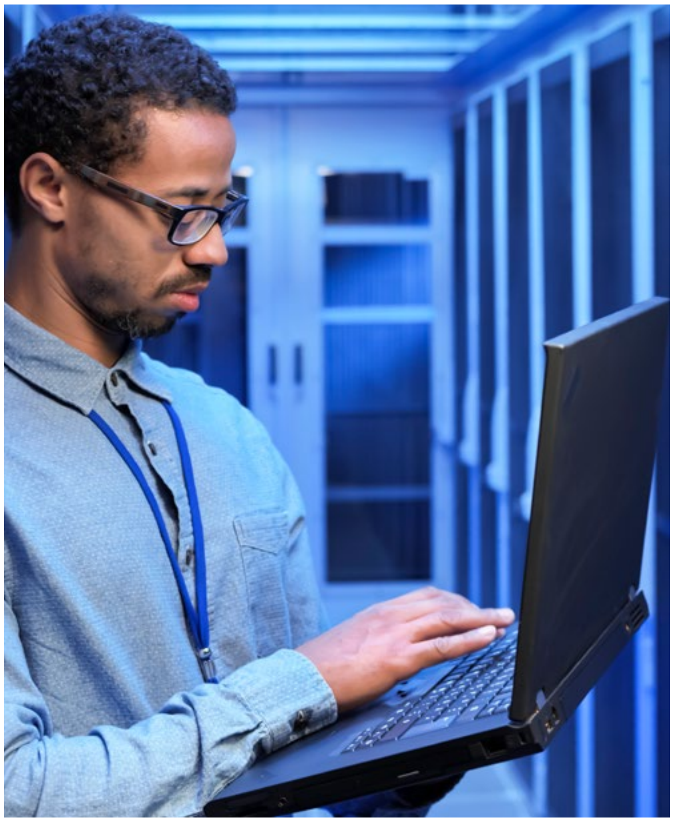
We need to address the above security challenges with microservice to avail the benefits offered by them. Below sections explain about the edge security and how our microservices can be protected and exposed by the API (Application Programming Interface) gateway and service mesh.

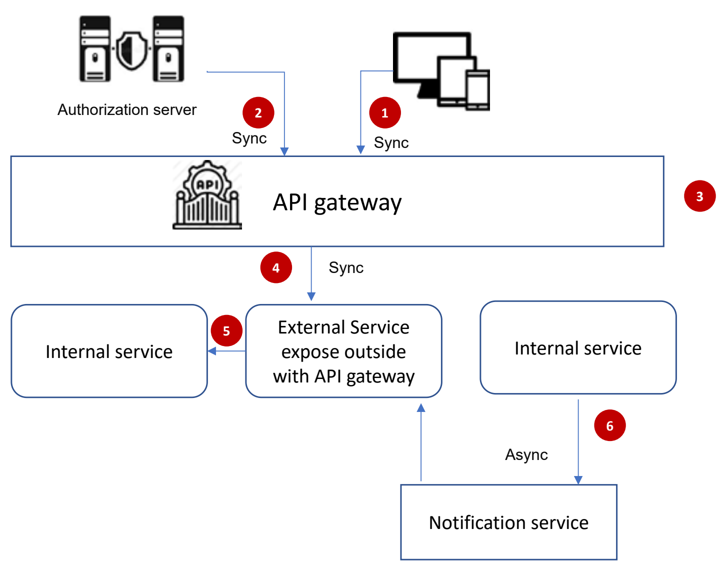
Edge Security

Edge security refers to protecting the data that resides and moves away from the protected boundaries of a centralized data center or cloud. We can use API gateway at the edge to protect and expose our microservices. Microservices are accessed via an API gateway by the client/front end application. The API gateway handles the security, throttling, versioning, traffic management, and analytics.

API Gateway

An API gateway is used to expose the microservice to client applications or services. External microservices that need to be accessed outside are exposed via an API gateway. Basic authentication and mutual TLS (Transport layer security) are required to secure microservices. OAuth 2.0 is the standard way to secure microservices at the edge. There are five different main grant types available Client credentials, Resource owner password, Authorization code, Implicit and Refresh token. Based on the application type, we need to choose the appropriate grand type. For an example we can use client credentials for authentication between two system without user interaction. OAuth 2.0 allows to add grant types as needed. A reference token and a JSON web token are the two types of access tokens. In the case of a reference token, it needs to be verified by the issuer's authorization server. If it is a JSON web token, then the gateway performs validation. Communication between gateway and microservices is secured by security system rules or mutual TLS (Transport layer security) or both. the mutual TLS and user context-protected communication between two internal services, which spread to other internal services. Asynchronous communication with internal services can be achieved with notification services.



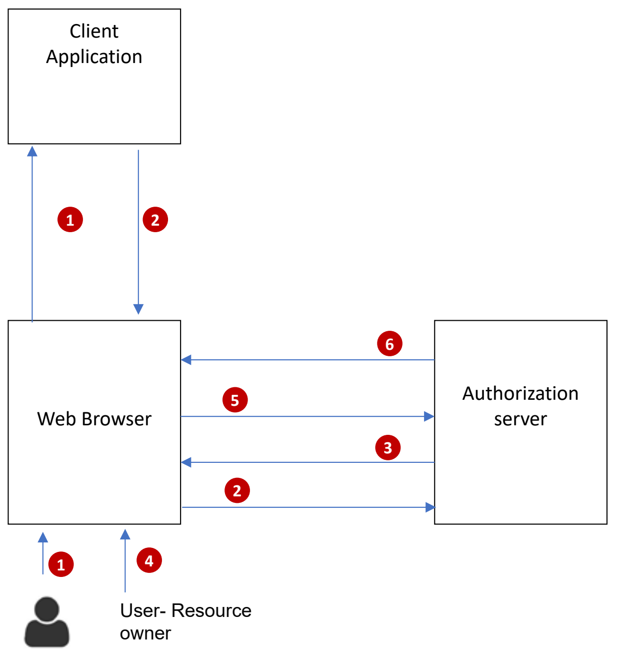


1. Systems or Application accessing APIs (Application Programming Interfaces) on behalf of themselves or human or another system
2. Authorization server with OAUth2
3. An API gateway which performs security screening, throttling, and publishing analytics to analytical server
4. Passing the end user context to the upstream service with HTTP header or with JWT
5. Service to service communication protected with mutual TLS and propagate user context one service to the other service
6. Asynchronous communication between two services can be achieved with notification service as the message broker

Key players in API management are Google Apigee, Kong API gateway, TYK API gateway, IBM API connect, Oracle API gateway, MuleSoft, Microsoft, Software AG, Axway, TIBCO Software and WSO2

SPA (Single page Application) OAuth 2.0 and OpenID connect

Users access the client application URL (uniform resource locator) from the web browser. The client application will redirect the user to the authorization server. The client application will send the client ID and redirect the application URL (uniform resource locator). The authorization server will prompt the user with a credentials page from the authorization server. After the user enters the credentials, the information is sent to the authorization server from the browser. The authorization server sent the access token back to the client application.

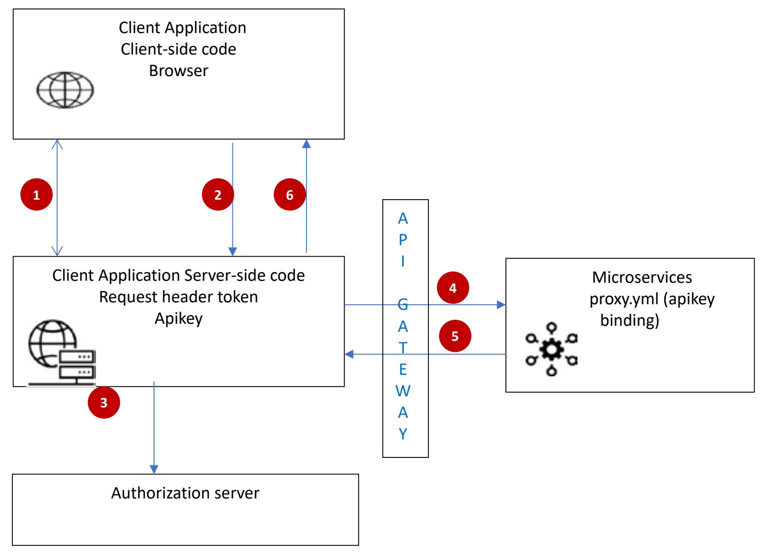


1. User enters URL of client application in web browser
2. Client application will redirect user to Authorization server
3. Authorization server prompt for credentials
4. User enter credentials
5. Browser post credential to Authorization server
6. Authorization server sends the access token

Client application server-side and client-side code

In most cases, single-page applications are used as client applications. Even though we call it a client application, it has two sections of code: client and server. Client-side code is sent from the web app server to the client browser as a bundle and gets executed by the browser. Server-side code resides on the web app server and gets executed there. One of the important responsibilities of server-side code is to act as a proxy service for the backend microservice running behind the API gateway.

Running this proxy service on a web application server adds more security for the application as the backend microservices are not called directly from the client browser. Always from the client browser, the proxy services on the web app server are called from client-side code with local host URLs (uniform resource locators). While calling this proxy service, the OAuth 2.0 token was passed in the header. Further, on the web app server-side, this token is validated with the authorization server before calling the backend microservices.



1. Client browser accessing the application URL the static contents are sent from web server
2. Client browsers will store the token they received in the app state after access it. Backend microservices are not directly called from the client browser; instead, it will call a proxy service running on the web server with this token.
3. Server-side service proxy will call the backend microservice with the apikey stored on the webserver after. OAuth 2.0 token is validated with authorization server
4. Apikey from the webserver proxy service request will be validated in Api gateway and allowed to access the backend service endpoint
5. Microservices endpoint sends response to the proxy service
6. Webserver proxy service send the response to client browser and stored in app state

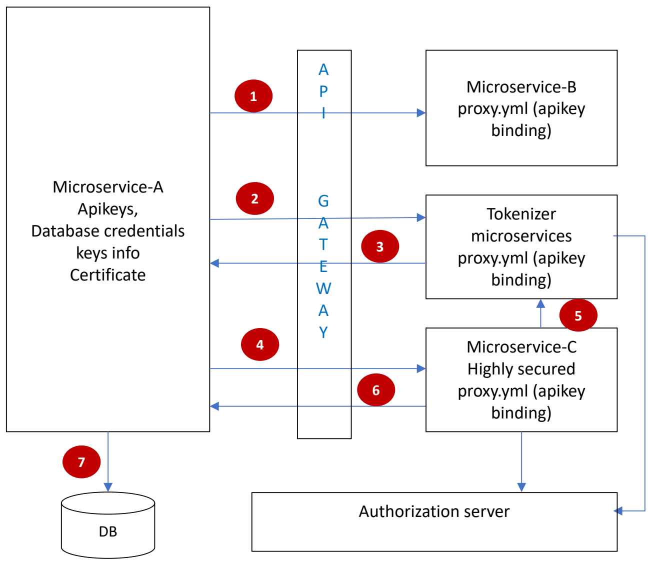
Secure microservices to microservices communication with API Key and Database access with certificate

An API key can be used to secure microservice-to-microservice communication. It is an identity assigned to a client application by microservices. The API key is used by the client applications in the request while calling the microservices. Microservices will validate this API key for authentication and authorization and either allow access to the end points or deny it.

Microservice-A wants to access or communicate with Microservice-B endpoints, then it will make a call with an API key in the header. The API key can be stored and accessed from the container cluster management configuration secret specific to the environment. Microservice-B will have the proxy configuration for that same API key, which is passed from Microservice-A and allows access to Microservice-B.

A defense-in-depth strategy is used to secure sensitive services with additional security layers. Microservice-C is a sensitive service with a high security requirement, and Microservice-A needs to access it. For added security, the tokenizer service is used to provide a service source token. This tokenizer service is an additional service that will generate and maintain the lifecycle of the custom service source token. The service source token will be specific to the transaction and valid for one time. Initially, Microservice-A will call the tokenizer service with the access token. This access token will be validated on the authorization server, and after that, the tokenizer service will provide a service source token. This additional service source token will be sent in a payload to Microservice-C along with the API key and access token. After validating the service source token and access token, a highly secure service will respond with data or send an error message.

Microservice-A can access database with a certificate and credentials. The certificate can also be installed dynamically from the secure location or be installed during the initialization script. Credentials can be accessed from the cluster management configuration secret section.



1. Microservice-A calling another Microservice-B with API Key which it will get dynamically from the cluster management configuration secret section. This API key would be added in API proxy with API key binding on another service to allow access
2. Microservice-A calling another Highly secure Microservice-C. Then tokenizer services will be adding more security by providing service source token. Which needs to provide during access High secure microservice along with access token additionally
3. Tokenizer services provide service source token after validating the access token
4. After receiving services source token from tokenizer service. Microservice-A calls the highly secure Microservice-C with service source token in the payload along with access token and API key in header
5. Highly secure Microservice-C check the service source token in the payload with tokenizer service and validate along with the API Key and access token validation
6. Highly secure Microservice-C send response after the validation of access token, apikey, service source token
7. Microservices connect to database with the dynamically installed security certificates with TLS connection along with database credentials

Service to service communication with service mesh

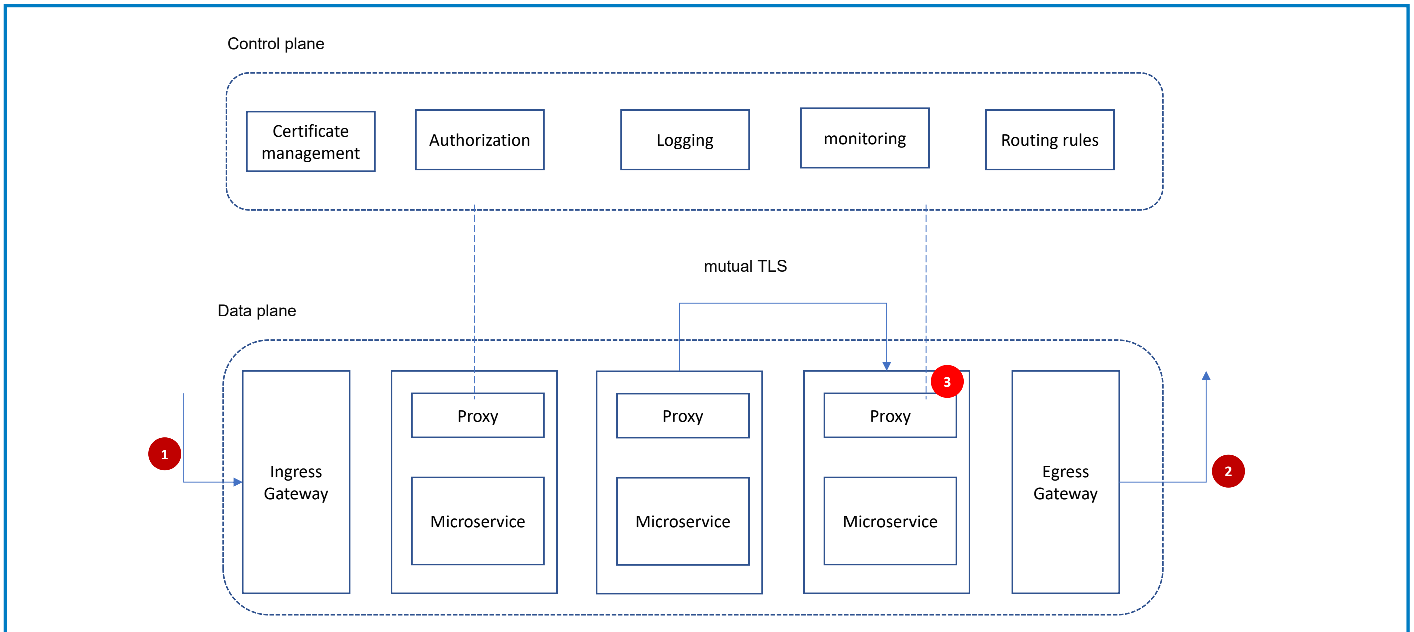
Service-to-service internal communication is done through service mesh. Data plane and control plane make up the architectural pattern known as "service mesh." These planes use abstraction to isolate microservices from routing, security, observability, and resilience. Along with the microservices, an in-service mesh service proxy will be deployed to intercept traffic to and from the ingress and egress gateways and enforce security. Service-to-service communication can be secured by mutual TLS (Transport Layer Security).

Services receiving calls from external applications or external services are secured by API gateways. Service-to-service internal communication is done through service mesh. Data plane and control plane make up the architectural pattern known as service mesh. These planes use abstraction to isolate microservices from routing, security, observability, and resilience. Along with the microservices, an in-service mesh service proxy will be deployed to intercept traffic to and from the ingress and egress gateways and enforce security. Service-to-service communication can be secured by mutual TLS (Transport Layer Security).

Data planes consist of three main components: a service proxy, an ingress gateway, and an egress gateway. All the requests and responses will go through the service proxy for the microservice. It will be responsible for security, monitoring, traffic, service discovery, and a circuit breaker to support resilience for all inbound and outbound traffic. All the traffic entering the service mesh goes through the ingress gateway, which will identify and dispatch the request to the appropriate service proxy. All the traffic leaving the service mesh goes through the egress gateway.

Control planes in the service mesh act as a policy administration point. It consists of the control instruction for the service proxies.

The service mesh architectural pattern has multiple implementations, like Istio, HasiCrop Consul, AWS (Amazon Web Services) app mesh, Azure service fabric, and Red Hat OpenShift service mesh. Istio is an open-source, popular service mesh created by Google that is widely used for service mesh implementation.



1 All the incoming request to service mesh goes via ingress gateway

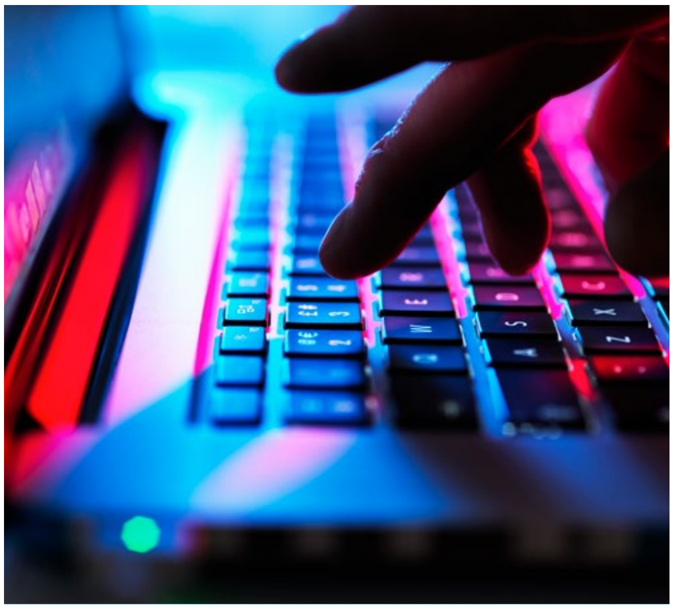
2 All the call outside from service mesh goes through the Egress Gateway

3 Service proxy plays the role of policy enforcement point

Microservices Security Patterns

Security patterns are required to minimize the vulnerability of the system. Following below patterns would reduce the security threat and attacks from various vectors.

Secure by design: Follow secure code practices to sanitize the user input. User inputs should be validated on both the client and the server sides. Especially string data with length validation and whitelist validation.



Scan dependencies: Scan the dependencies with libraries and packages for vulnerabilities as part of your delivery pipeline. Consider adding a bot like dependabot to automatically scan the repository for vulnerable packages or libraries and then remediate with a pull request.

Use HTTPS (Hypertext Transfer Protocol Secure): Use the certificate for all the sites, even he static content sites. The communication protocol is encrypted using TLS (Transport Layer Security) or SSL (Secure Sockets Layer). It authenticates the accessed web site with the certificate provider. protected data integrity and privacy during transit.

Secure with access and an identity token: OAuth 2.0 is an authorization framework, and OpenID Connect is the identity layer of OAuth 2.0. There are four main actors who play distinct roles in the access delegation. Resource servers host resources and ensure accessibility based on certain conditions. The client is a resource consumer. The end user is the owner of the resources, and the authorization seraver is the one that issues access tokens. It can be a reference token or a JSON Web Token, or it can be PASETO (platform agnostic security token)

Encrypt and protect secrets: Store secrets like API keys and credentials in environment variables. Use a vault services like AWS KMS or Azure Key Vault.

Security in the delivery pipeline: Use DevSecOps to inject security into the delivery pipeline. Use containers, cluster management lint and scanning, static analysis security testing, and dynamic analysis security testing in pipeline steps.

Conclusion

Microservices bring greater value and benefits, and the need for security for these services also increases. The security patterns and strategies, like isolation, defense in depth, and DevSecOps, help improve the security of microservices. Based on our needs and level of security requirements, we must formulate a microservice security strategy with an API gateway and/or service mesh.

Recommended approach to secure microservices

- Implement OAuth 2.0 and OpenID Connect to secure microservices at the edge.
- Use mutual TLS (Transport Layer Security) and ensure traffic is secure between services.
- Implement an API gateway for exposing the microservices to the outside world and for managing incoming traffic from external services and applications.
- For internal traffic service-to-service communication, use service mesh.
- Define the DevSecOps delivery pipeline to perform various security scanning steps for the microservices, like static code analysis, container scanning, and cluster management scanning.

References

1. Microservice Architecture https://microservices.io/
2. Microservices Security in Action https://www.oreilly.com/library/view/microservices-security-in/9781617295959/
3. How to Secure Microservices Architecture https://securityintelligence.com/posts/how-to-secure-microservices-architecture/
4. Microservices Security: Challenges and Best Practices https://www.aquasec.com/cloud-native-academy/cloud-native-applications/microservices-security/

About the Author

Saravana Krishnan Palani

Senior Technology Architect

About the Mentors

Jitendra Jain

Principal Technology Architect;

CLOUD PROFESSIONAL

Amit Chandra Kesh

Senior Technology Architect

Partha Konwar

Senior Technology Architect

For more information, contact askus@infosys.com

Infosys®
Navigate your next

Infosys.com | NYSE: INFY

Stay Connected