Secure Microservices

Arnav Gupta

December 11, 2024

Contents

1 Challenges

Involves many small services, each handling specific business.

Concerns involve access control, secure communication, protection against data theft, or unauthorized access.

Goal: controlling access and security communication routes

2 Protecting Service Mesh

Objective is to protect internal services from public access:

- minimize attack surface
- prevent direct public access to services

Inadequate protection can result in:

- increased security overhead
- easier target identification for attackers
- potential for escalated attacks upon service compromise

Helps to use API gateway:

- defines exposed routes to client applications
- differentiates between public and internal APIs
- reduces attack surface
  - expose only necessary routes and endpoints
  - minimize potential targets for attackers

To implement security using API gateway:

- global auth techniques: enforce API access tokens for auth and verify pre-shared secrets for each client app
- differentiation of app access: unique API access token for each app, which helps in limiting API access based on client app
- security strategies: implement rate-limiting to prevent DoS attacks and mitigate overload of microservices by external requests
- management security: enforce strict access rules to manage infrastructure and integrate with active directory or simple services for control plane protection
- protecting outbound communication: secure communication with external dependencies

3 Communication Protection: External

With certification verification, can prevent man-in-the-middle attacks by ensuring that client communicates with intended server (can distinguish between malicious and legitimate servers).

3.1 TLS

Use of HTTPS/TLS encrypts and authenticates communication channels and ensures confidentiality and authenticity of data in transit.

TLS protocol involves a handshake between communicating parties and server presents a certificate containing a public key.

TLS Certificate components:

- public key enclosed in signed envelope
- signature by issuing authority using a private key

Private key security prevents leakage of private keys that compromises certificate authenticity.

Client certificate verification strategies:

- accept all certificates
- accept self-signed certificates
- accept certificates signed by trusted authorities
- accept only a specific certificate (certificate pinning)

In standard TLS, only client verifies server certificate.

In **mutual TLS**, both client and server present and verify certificates using similar verification strategies as standard TLS.

4 Communication Protection: Internal

Want to avoid different services talking directly to one another.

Must limit damage that can be done in scenario when intruder breaks into cluster.

Have communication be between namespaces, grouping clusters into smaller, manageable units, where each namespace acts like a distinct segment.

Network policies in Kubernetes:

- regulate traffic between namespaces
- define rules for inbound and outbound traffic

Outbound traffic control prevents misuse of services for attacks like server-side request forgery (SSRF) and protects against installation of malicious software.

Behaviour monitoring frameworks define expected behaviour via config files.

Security alerts notify operators on deviation from normal behaviour and enhance detection of suspicious activities within the cluster.

Resource quotas set limits on resources like memory and CPU per namespace.

Preventing DoS attacks by limiting resource consumption (essential for maintaining service availability).

5 Individual Microservice: Interfaces

5.1 JWTs

Use JSON WebTokens (JWTs) for identity verification:

- consists of header, body, and signature
- header specifies signing algorithm
- body contains user claims and token validity
- signature ensures token integrity and authenticity

Balance token lifespan for security and validity. Use refresh tokens for longer validity.

User credentials verified by auth service and successful validation leads to JWT issuance.

To manage tokens, issue both access and refresh tokens. Revoke and renew tokens upon expiration.

To minimize risks of token theft, automatically invalidate all user tokens if one is compromised.

For token management in microservices:

- use external provider for token generation and have token validation at API gateway or individually by services
- (undesirable) each service manages its own token generation and validation
- centralized token generation, so individual services validate tokens using asymmetric cryptography with secret key only known to auth service

5.2 Port Management

Minimize open ports (only essential ports open) and avoid unnecessary management interfaces.

Additional open ports increase attack surface (high-risk targets).

5.3 Access Control

Selective visitor acceptance: use access-control lists for critical services

API gateway enriches HTTP headers with source IP and services check IP against allow list for access.

5.4 Info Leakage

Prevent leakage of internal backend details by sanitizing errors and exceptions.

5.5 Middleware

Using middleware can cross-cut security concerns. Recommended order for middleware is exception, authentication, authorization, and access control.

Can also have rate-limiting middleware based on source IP or user identity.

6 Individual Microservice: Internals

Docker container hardening: restrict ports for communication and avoid running containers as root user (create new user in Dockerfile, dropping unnecessary rights, file system read-only)

Input data validation is essential to prevent unauthorized data manipulation.

Risks of input data validation:

- SQL injection risk: risk from string concatenation in SQL queries
- remote code execution (RCE) vulnerability: potential for attacker-controlled process execution or reverse shell creation
- cross-Site Scripting (XSS): data passed between services without sanitization, so risks in frontend usage

To validate input data: validate integer ranges, string usage contextually, and ensure data transfer object deserialization is secure.

Can also do context-specific string validation.

Can store secrets with hard coding, providing secrets as files or environment variables, or dynamically retrieving secrets via a network (read only vault that checks IP).

Key revocation process and key rotation are important.