Microservices Security

Bảo mật luôn là một yếu tố cốt lõi trong việc xây dựng và triển khai các hệ thống phần mềm, đặc biệt là đối với kiến trúc microservices. Trong phần mở đầu của chương "Microservices Security" thuộc khóa học *Master Microservices with Spring Boot, Docker, Kubernetes*, người học được giới thiệu về thách thức lớn thứ 9 trong hành trình xây dựng microservices: đó là bảo mật hệ thống. Tác giả nhấn mạnh rằng, cho dù hệ thống của chúng ta có được phát triển tốt đến đâu, nếu không có một cơ chế bảo mật chặt chẽ, thì dữ liệu nhạy cảm rất dễ bị lộ và gây hậu quả nghiêm trọng cho tổ chức. Việc các microservice có thể bị gọi bởi bất kỳ ai, thậm chí từ những người dùng không được xác thực, là một vấn đề đáng lo ngại và không thể chấp nhận trong môi trường sản xuất thực tế.

Khái niệm về bảo mật trong microservices không chỉ đơn thuần là ngăn chặn truy cập trái phép, mà còn bao gồm cả việc đảm bảo đúng người, đúng quyền mới được phép truy cập tài nguyên phù hợp. Điều này dẫn đến hai thành phần cốt lõi là xác thực (authentication) và phân quyền (authorization). Trong đó, xác thực nhằm kiểm tra danh tính người dùng hoặc hệ thống đang cố gắng truy cập dịch vụ, còn phân quyền sẽ quyết định xem người dùng đã xác thực có đủ quyền để thực hiện hành động cụ thể hay không.

Một điểm đáng chú ý được đưa ra là: thay vì tích hợp bảo mật vào từng microservice riêng biệt, điều sẽ gây khó khăn trong việc quản lý và bảo trì về lâu dài, ta nên xây dựng một cơ chế quản lý định danh và truy cập tập trung (IAM). Giải pháp được đề xuất bao gồm việc sử dụng các tiêu chuẩn bảo mật phổ biến như OAuth2, OpenID Connect, kết hợp với công cụ Keycloak và Spring Security – một framework chuyên biệt của Spring trong việc xử lý các yêu cầu bảo mật. Những công cụ và tiêu chuẩn này sẽ giúp đơn giản hóa việc triển khai bảo mật, đồng thời đảm bảo tính linh hoạt và nhất quán trên toàn bộ hệ thống microservices.

Vấn đề của Basic Authentication và nhu cầu sử dụng OAuth2

Trong các hệ thống hiện đại, việc xác thực người dùng là một yêu cầu bắt buộc. Trước đây, Basic Authentication là một trong những phương pháp xác thực phổ biến, trong đó username và password của người dùng được gửi trong mỗi request. Tuy nhiên, phương pháp này bộc lộ nhiều nhược điểm nghiêm trọng:

- Gắn chặt giữa client và authentication server (tight coupling): Không có sự phân tách rõ ràng, dẫn đến khó mở rộng và thay thế.

- Không hỗ trợ bên thứ ba (third-party authorization): Không có cách nào để ứng dụng của bên thứ ba truy cập tài nguyên của người dùng mà không cần lưu trữ mật khẩu của họ.
- Rủi ro bảo mật cao: Mỗi lần truy cập đều gửi credentials, nếu bị lộ sẽ ảnh hưởng nghiêm trọng đến tài khoản người dùng.
- Không kiểm soát được quyền truy cập cụ thể: Không có cơ chế cấp quyền một phần cho các ứng dụng bên thứ ba (chỉ có "all or nothing").

OAuth2 – Giải pháp xác thực và phân quyền hiện đại

OAuth2 ra đời để giải quyết các vấn đề trên. Đây là một protocol (giao thức) xác thực và phân quyền hiện đại, giúp:

- Phân tách server xác thực (Authorization Server) ra khỏi các dịch vụ khác (như Resource Server), từ đó giảm coupling.
- Cho phép cấp quyền truy cập giới hạn cho bên thứ ba mà không cần chia sẻ password.
- Hỗ trợ token-based authentication, hạn chế rò rỉ credentials và tăng bảo mật.
- Dễ dàng tích hợp với các hệ thống khác như Google, Facebook, GitHub,... mà không cần quản lý người dùng nội bộ.

Ví dụ thực tế minh họa:
Người dùng đăng nhập StackOverflow bằng tài khoản GitHub, trong đó:

- StackOverflow là Client Application
- GitHub là Authorization Server
- Người dùng cấp quyền truy cập thông qua một access token mà không cần đưa mật khẩu GitHub cho StackOverflow.

Các thành phần chính (Jargons / Terminologies) trong OAuth2

OAuth2 định nghĩa một số vai trò (roles) quan trọng sau:

Role	Chức năng
Resource Owner (User)	Chủ sở hữu tài nguyên (người dùng)
Client Application	Ứng dụng bên thứ ba muốn truy cập tài nguyên
Authorization Server	Cấp token truy cập (Access Token), thực hiện xác thực người dùng

Resource Server	Server cung cấp tài nguyên khi có token hợp lệ
Access Token	Token tạm thời cho phép client truy cập tài nguyên
Refresh Token	Token để lấy lại access token mới khi token cũ hết hạn

Sự khác biệt giữa Authentication và Authorization

- Authentication (xác thực): Là quá trình kiểm tra danh tính của người dùng – người đó là ai.
- Authorization (ủy quyền): Là quá trình kiểm tra xem người dùng đã được cấp quyền để truy cập tài nguyên nào đó chưa.

Trong khi OAuth2 là một framework hỗ trợ authorization (ủy quyền) – cho phép bên thứ ba truy cập tài nguyên thay mặt người dùng – thì OAuth2 không cung cấp thông tin để xác thực danh tính người dùng. Điều này khiến OAuth2 không phù hợp để dùng cho các luồng đăng nhập (login flows).

OpenID Connect ra đời để giải quyết điểm yếu của OAuth2

- OpenID Connect (OIDC) là một authentication layer được xây dựng trên nền của OAuth2.
- Nó mở rộng OAuth2 bằng cách giới thiệu ID Token – một token định danh người dùng được phát hành sau khi xác thực thành công.
- ID Token thường được mã hóa dưới dạng JWT (JSON Web Token) và chứa các thông tin chuẩn hóa như: sub (user id), name, email, iat (issued at), exp (expiry), v.v.

Tại sao OpenID Connect quan trọng

- Cho phép xác thực người dùng tập trung thông qua một identity provider (IdP).
- Hỗ trợ đăng nhập một lần (Single Sign-On - SSO) cho nhiều ứng dụng.
- Chuẩn hóa thông tin người dùng giữa các hệ thống.
- Phù hợp với các hệ thống hiện đại cần cả xác thực và ủy quyền tách biệt.

Ví dụ thực tế

- Khi đăng nhập vào StackOverflow thông qua GitHub, ta không cần chia sẻ username/password GitHub cho StackOverflow. Thay vào đó, StackOverflow sẽ sử dụng **OAuth2** để ủy quyền, và **OIDC** để xác thực người dùng, nhận về **ID Token** chứa thông tin như tên và email để hiển thị và ghi nhận đăng nhập.

Identity and Access Management – IAM

Được xây dựng dựa trên các tiêu chuẩn **OAuth2** và **OpenID Connect**. Đây là những **specification (đặc tả)** định nghĩa cách xây dựng các hệ thống bảo mật trong ứng dụng web hiện đại. Tuy nhiên, chỉ dựa vào các đặc tả này thì không thể tự xây dựng một hệ thống bảo mật hoàn chỉnh – cần có các **sản phẩm phần mềm cụ thể** để triển khai thực tế.

Lý do cần IAM Products

- Các tiêu chuẩn như OAuth2 và OpenID Connect chỉ cung cấp định hướng kỹ thuật, chứ không phải là phần mềm có thể sử dụng ngay.
- Việc xây dựng hệ thống xác thực và ủy quyền từ đầu là rất phức tạp và tốn kém, đặc biệt với các tổ chức vừa và nhỏ.
- Vì vậy, nhiều công ty đã tạo ra các IAM products dựa trên các tiêu chuẩn này nhằm phục vụ nhu cầu triển khai bảo mật dễ dàng và chuẩn hóa.

Giới thiệu Keycloak

- Keycloak là một sản phẩm mã nguồn mở (open-source) về Identity và Access Management.
- Nó được tài trợ bởi Red Hat và hỗ trợ các tiêu chuẩn phổ biến như:
    - OpenID Connect
    - OAuth2
    - SAML
- Các tính năng đáng chú ý:
    - Đăng nhập một lần (Single Sign-On - SSO)
    - Xác thực mạng xã hội (Social Login)

Keycloak cho phép triển khai nhanh chóng một Authorization Server mà không cần phát triển từ đầu, phù hợp với các doanh nghiệp vừa và nhỏ, cũng như cá nhân học tập và nghiên cứu.

So sánh với các sản phẩm IAM khác

Ngoài Keycloak, còn có nhiều sản phẩm thương mại được xây dựng trên nền OAuth2 và OpenID Connect như:

- **Okta**: phổ biến trong các ứng dụng doanh nghiệp lớn.
- **Amazon Cognito**: dịch vụ IAM có khả năng mở rộng cao, tích hợp chặt chẽ với AWS.
- **FusionAuth, Forgerock**: các giải pháp IAM thương mại khác.

Client Credentials Grant Type là gì?

- Dành cho **ứng dụng không có người dùng** (non-user-based apps), như: các backend services, APIs, servers...
- **Client (ứng dụng)** sẽ lấy access token từ **Auth Server (Keycloak)** bằng cách dùng client_id và client_secret.
- Sau đó sử dụng token này để gọi đến **resource server** (trong bài là Gateway Server).

Khi nào dùng Client Credentials Grant Type?

- Dùng khi **không có người dùng (end-user)** hoặc **UI application**.
- Áp dụng khi **2 ứng dụng backend (2 API/microservice)** cần giao tiếp với nhau một cách bảo mật.

Ví dụ:

- Hệ thống partner gọi API quản lý tài khoản.
- Backend server gửi dữ liệu đến hệ thống khác.

Kiến trúc hệ thống:

- Client Application: là các hệ thống backend bên ngoài muốn gọi API (ví dụ: hệ thống đối tác, backend của mobile app...).
- Auth Server (Keycloak): cấp access token cho client đã đăng ký.
- Gateway Server (Edge Server): là resource server, kiểm tra token trước khi chuyển tiếp request đến các microservices.
- Microservices: Accounts, Loans, Cards...

Keycloak là gì?

Keycloak là một Identity and Access Management (IAM) tool mã nguồn mở, dùng để:

- Xác thực (Authentication) người dùng và dịch vụ.
- Phân quyền (Authorization) truy cập vào các tài nguyên.
- Hỗ trợ các chuẩn bảo mật như OAuth2, OpenID Connect, SAML.

Nó đóng vai trò như một Authentication Server (Auth Server) trung tâm trong hệ thống microservices.

Tại sao dùng Keycloak làm Auth Server?

- Tránh tình trạng các microservice **phải tự xử lý đăng nhập, phân quyền**.
- Dễ dàng tích hợp với **các giao thức bảo mật chuẩn** (OAuth2, OpenID Connect).
- Quản lý tập trung người dùng, client, role và token.
- Có thể tích hợp **Single Sign-On (SSO)** giữa nhiều ứng dụng.

Cách Setup Auth Server bằng Keycloak

1. Cài đặt và khởi động Keycloak

- Cách đơn giản nhất là dùng Docker:

```
docker run -p 8080:8080 \
  -e KEYCLOAK_ADMIN=admin \
  -e KEYCLOAK_ADMIN_PASSWORD=admin \
  quay.io/keycloak/keycloak:latest start-dev
```

- Truy cập admin console tại: http://localhost:8080

2. Tạo Realm

- Realm là không gian riêng biệt trong Keycloak để quản lý user, client, roles.
- Bạn có thể dùng realm mặc định (hoặc tạo mới nếu cần).

3. Tạo và Cấu hình Client (Application)

- Mỗi ứng dụng (client) muốn giao tiếp với Auth Server đều cần được **đăng ký trước**.

- Trên Admin Console:
    - Truy cập **Clients** > **Create Client**
    - **Client Type**: OpenID Connect (bao gồm OAuth2)
    - **Client ID**: ví dụ easybank-callcenter-cc
    - **Bật**: Client Authentication
    - **Tắt**: Standard Flow, Direct Access Grant
    - **Bật**: Service Account Roles (để dùng **Client Credentials Grant Flow**)

4. Lấy Client ID và Secret

- Sau khi tạo client, vào tab **Credentials** để lấy:
    - client_id
    - client_secret
- Đây là thông tin mà **ứng dụng bên ngoài (external client)** cần cung cấp khi gọi Auth Server để lấy access token.

5. Auth Flow tương ứng

Với **Client Credentials Grant Flow**:

- Dùng khi **2 ứng dụng backend giao tiếp với nhau** (không có người dùng).
- Ứng dụng gửi client_id + client_secret đến /token endpoint của Keycloak để nhận về access_token.

Ví dụ thực tế

- Ứng dụng nội bộ EasyBank Call Center muốn gọi đến API Gateway.
- Trước tiên, nó gọi đến Keycloak Auth Server để lấy access token thông qua client_credentials grant.
- Sau đó, dùng token này để truy cập các microservice thông qua API Gateway.

Securing Gateway Server as a Resource Server

1. Cấu hình Gateway để xác thực JWT

Trước tiên, để Gateway hiểu và xác thực JWT token, ta thêm các dependencies sau vào pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-security-oauth2-jose</artifactId>
</dependency>
```

Sau đó cấu hình application.yml như sau:

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8181/realms/master
```

2. Cấu hình phân quyền chi tiết theo role

Tạo lớp SecurityConfig để chỉ định quyền truy cập cho từng endpoint qua Gateway

```
@EnableWebFluxSecurity
public class SecurityConfig {

  @Bean
  public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity http) {
```

```
    http
        .authorizeExchange(exchanges -> exchanges
            .pathMatchers("/eureka/**").permitAll()  // Cho phép Eureka tự do
            .pathMatchers(HttpMethod.GET, "/accounts/**").hasRole("USER")   // USER được gọi GET
            .pathMatchers(HttpMethod.POST, "/accounts/**").hasRole("ADMIN") // ADMIN được gọi POST
            .anyExchange().authenticated() // Các route khác phải xác thực
        )
        .oauth2ResourceServer(ServerHttpSecurity.OAuth2ResourceServerSpec::jwt);

    return http.build();
  }
}
```

Kết quả đạt được:

- Gateway giờ đây chỉ cho phép request hợp lệ đi qua nếu có JWT token đúng định dạng và chữ ký.
- Việc phân quyền rõ ràng giúp bảo vệ từng microservice khỏi truy cập trái phép.
- Mọi authentication & authorization đều được tập trung ở Gateway, giúp các service phía sau đơn giản hơn và tập trung vào xử lý nghiệp vụ.
- Đây là bước quan trọng để biến hệ thống microservices trở nên **bảo mật**, **linh hoạt** và **chuẩn hóa theo mô hình IAM (Identity & Access Management)**.

Quy trình Authorization Code Flow với Postman

1. Kiểm tra API không có token

- Gửi request đến API accounts → Nhận 401 Unauthorized.

2. Cấu hình OAuth2 trong Postman

- Vào tab Authorization:

- Type: OAuth 2.0
- Header Prefix: Bearer
- Configure New Token:
  - Token Name: auth_code_access_token (tuỳ bạn đặt)
  - Grant Type: Authorization Code
  - Callback URL: Postman tự dùng https://oauth.pstmn.io/v1/callback (không thay đổi được)
  - Auth URL: http://<KEYCLOAK_HOST>/realms/<REALM_NAME>/protocol/openid-connect/auth
  - Access Token URL: http://<KEYCLOAK_HOST>/realms/<REALM_NAME>/protocol/openid-connect/token
  - Client ID / Secret: Lấy từ client đã đăng ký trong Keycloak
  - Scope: openid email profile
  - State: chuỗi ngẫu nhiên (để chống CSRF)
  - Client Authentication: Send client credentials in body
  - Bật "Authorize using browser"

3. Lấy access token

- Nhấn Get New Access Token
- Trình duyệt mở trang đăng nhập Keycloak
- Đăng nhập bằng user thường (madan / 12345)
- Postman nhận authorization code → đổi lấy access token
- Nhấn Use Token

4. Kiểm tra truy cập sau khi có token

- Gửi lại request với access token vừa nhận.
- Nhận 403 Forbidden → do user chưa có role accounts.

5. Gán role trong Keycloak

- Đăng nhập Keycloak với tài khoản admin.
- Vào mục Users → chọn user madan
- Vào tab Role Mappings
- Gán các roles: accounts, cards, loans

6. Lặp lại lấy token và gọi API

- Đóng hết trình duyệt (xóa session admin).
- Lấy lại token bằng user madan.
- Gọi lại các API: accounts, cards, loans → thành công

Kết quả cuối cùng

- Gọi API thông qua Gateway thành công với token từ Authorization Code Flow.
- Các roles phân quyền hoạt động đúng.

Demo

Step 1: Thiết lập Keycloak

- Tạo Realm: springboot-microservices-realm
- Tạo Client: gateway-client (sử dụng confidential, enable authorization code grant)
- Tạo User: john (password: password)
- Cấu hình các roles và mappers cần thiết để đưa thông tin user (ID token, access token)

Step 2: Cấu hình Spring Cloud Gateway làm Resource Server

- Thêm dependencies:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Cấu hình application.yml:

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://keycloak:8080/realms/springboot-microservices-realm
```

Tạo lớp cấu hình bảo mật:

```
@EnableWebFluxSecurity
public class SecurityConfig {
    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity http) {
        return http
            .authorizeExchange(exchanges -> exchanges
                .pathMatchers("/eureka/**").permitAll()
                .anyExchange().authenticated()
            )
            .oauth2ResourceServer(oauth2 -> oauth2.jwt())
            .build();
    }
}
```

Step 3: Docker Compose

- Cấu hình file docker-compose.yml để chạy tất cả container:

```
version: "3.8"
services:
  keycloak:
    image: quay.io/keycloak/keycloak:latest
    command: start-dev
    ports:
      - "8080:8080"
    environment:
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
  gateway:
    build: ./gateway
    depends_on:
      - keycloak
    ports:
      - "8083:8083"
    environment:
      SPRING_PROFILES_ACTIVE: docker
```

Các services như accounts, loans, cards, eureka, configserver cũng được định nghĩa tương tự.