Security

Bu kategorideki her dersin şimdiye kadar varsaydığı koşulsuz güveni kapatmak: Spring Security'yi projeye ilk kez tanıtmak, authentication vs authorization ayrımı, JWT'nin kendi kendine yeterli doğası, api-gateway ve order-service'te AYRI AYRI JWT doğrulaması (zero trust), role-bazlı authorization, ve kimliğin correlation id'ye paralel şekilde servisler arası yayılması.

Orta 26 dk

Son güncelleme: 23 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/security

---

Security

Bu kategorideki her ders şimdiye kadar order-service ve inventory-service'in kendilerini çağıran her şeye basitçe güvendiğini varsaydı. Odak başka yerdeyken makul bir sadeleştirmeydi -- ama gerçek bir sistemin, bu kursun şimdiye kadar hiç değinmediği iki soruya cevap vermesi gerekir: bu isteği yapan kim, ve istediklerini yapmasına izin var mı? Bu ders, Spring Security'yi kursa ilk kez tanıtıyor, özellikle bu kategorinin servisleriyle sınırlı olarak.

Bir Mikroservis Sistemi İçin Security Ne Demek?

Tek bir uygulamada, security genellikle ön kapıda tek bir giriş kontrolü demektir. Bir mikroservis sisteminde, isteği alan HER servisin -- yalnızca herkese açık internete bakanın değil -- "bu kim, ve bunu yapmasına izin var mı" sorusuna kendi cevabı olması gerekir, çünkü bir istek, yanlış yapılandırma yüzünden ya da bilinçli olarak, herkese açık olana (api-gateway) hiç dokunmayan yollardan bir iç servise (inventory-service) ulaşabilir.

Neden Var?

Herhangi bir kimlik kontrolü olmadan, order-service'in ağına ulaşabilen HERHANGİ bir istek sipariş verebilir, ve inventory-service'e doğrudan ulaşabilen herhangi bir istek api-gateway'i tamamen atlayabilir -- api-gateway'in sağladığı yönlendirme ve sistem geneli konular (API Gateway dersine bakınız) kolaylık ve yapıdır, tek başlarına bir güvenlik sınırı değil. Yalnızca api-gateway'de kimlik kontrolü yapıp sonra her iç çağrıya koşulsuz güvenen bir sistem, yalnızca EN AZ korunan iç yolu kadar güvenlidir.

Tarihçe

HTTP API'leri için token-tabanlı kimlik doğrulama, tek sayfa uygulamaları ve mobil istemciler 2010'lar boyunca sunucu-render edilmiş, session-cookie-tabanlı girişlerin yerini aldıkça baskın desen haline geldi -- herhangi bir servisin bağımsız olarak doğrulayabildiği stateless bir token, paylaşılan bir sunucu-taraflı session'ın hiçbir zaman uyamayacağı kadar dağıtık bir sistemin şekline uyar. RFC 7519'da (2015) standartlaştırılan JSON Web Token (JWT), tam olarak kendi kendine yeterli ve bağımsız olarak doğrulanabilir olduğu için ("JWT: Kendi Kendine Yeterli, Doğrulanabilir Bir Kimlik" bölümüne bakınız) o token için en yaygın şekil haline geldi -- hiçbir servisin, bir isteği yapanın kim olduğunu kontrol etmek için merkezi bir session deposuna geri çağrı yapmasına gerek yok.

Authentication vs Authorization: İki Farklı Soru

Bu iki kelime genellikle gevşek kullanılır, ama gerçekten farklı sorulara cevap verirler. Authentication "bu kim?" diye sorar -- bir kimliği doğrulamak, tipik olarak bir token'ın imzasını kontrol ederek. Authorization "BU kimliğin BUNU yapmasına izin var mı?" diye sorar -- authentication başarılı olduktan SONRA gerçekleşen tamamen ayrı bir karar ("Authorization: Bir Endpoint'i Role Göre Kısıtlamak" bölümüne bakınız). Bir istek authenticated (gerçek, geçerli bir kimlik) olabilir ve yine de unauthorized olabilir (o kimliğin istediğini yapmasına yalnızca izin yok).

JWT: Kendi Kendine Yeterli, Doğrulanabilir Bir Kimlik

Bir JWT kendi claim'lerini (kim yayınladı, kimi tanımlıyor, hangi rolleri ya da scope'ları veriyor, ne zaman süresi doluyor) ve bunların hepsi üzerinde kriptografik bir imza taşır -- yayınlayanın public key'ine sahip herhangi bir servis imzayı doğrulayabilir ve claim'lere güvenebilir, BU spesifik kontrol için yayınlayana hiç doğrudan bağlanmadan. Bu, api-gateway ve order-service'in ikisinin de AYNI token'ı bağımsız olarak doğrulayabilmesini sağlayan şey ("Gateway'de Bir JWT'yi Doğrulamak" ve "Gateway Tek Başına Neden Yetmiyor" bölümlerine bakınız).

Gateway'de Bir JWT'yi Doğrulamak

api-gateway, dış bir isteği gören ilk servis, bu yüzden hiç geçerli token taşımayan bir isteği reddetmek için doğal ilk yer.

```
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.web.reactive.EnableWebFluxSecurity;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.web.server.SecurityWebFilterChain;

// api-gateway's FIRST security configuration -- this course has used no Spring
// Security anywhere until now (the AI ingestion endpoint in the quiz feature
// used a hand-written X-Api-Key check specifically BECAUSE Spring Security
// wasn't already a dependency, see that feature's own design notes). Spring
// Cloud Gateway runs on WebFlux (see the API Gateway lesson), so this uses the
// REACTIVE security config style (@EnableWebFluxSecurity, ServerHttpSecurity),
// not the servlet-based style order-service uses in OrderServiceSecurityConfig.
@EnableWebFluxSecurity
class ApiGatewaySecurityConfig {

    @Bean
    SecurityWebFilterChain securityFilterChain(ServerHttpSecurity http) {
        return http
                .authorizeExchange(exchanges -> exchanges
                        .pathMatchers("/actuator/health").permitAll()   // health
                                                                          // checks stay
                                                                          // public --
                                                                          // load
                                                                          // balancers
                                                                          // need to
                                                                          // reach them
                                                                          // unauthenticated
                        .anyExchange().authenticated())
                // oauth2ResourceServer + jwt() tells Spring Security that a valid
                // request carries a JWT in its Authorization header, and how to
                // VERIFY it (see ApiGatewayJwtConfig.yml for where the verification
                // key comes from) -- see "JWT: A Self-Contained, Verifiable Identity".
                .oauth2ResourceServer(oauth2 -> oauth2.jwt(jwt -> {}))
                .build();
    }
}
```

```
# Added to api-gateway's application.yml. Spring Security's resource server
# support needs to know WHERE to fetch the public key(s) it uses to verify a
# JWT's signature -- issuer-uri points at an identity provider (a dedicated
# authentication service; this course assumes one already exists and issues
# tokens, the same way it assumes a Kafka broker already exists for the
# Event-Driven Architecture & Kafka lesson -- building an identity provider
# itself is out of scope here).

spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://auth.example.com/   # Spring Security fetches this
                                                    # issuer's public signing keys
                                                    # automatically on startup --
                                                    # no key material is
                                                    # hardcoded anywhere in
                                                    # api-gateway's own config
```

> Bu kurs, bu derse kadar hiçbir yerde Spring Security kullanmadı -- bu projenin quiz özelliği için inşa edilen AI ingestion endpoint'i, bunun yerine bilinçli olarak elle yazılmış bir `X-Api-Key` kontrolü kullandı, özellikle tek bir internal endpoint için tüm bir security framework'ü eklemek orantısız olacağı için. Gerçek kullanıcı kimliğini işleyen, herkese açık bir gateway, Spring Security'yi getirmeyi haklı çıkaran tam olarak bu türden bir durum.

Gateway Tek Başına Neden Yetmiyor: Servisler Arasında Zero Trust

order-service kendisine ulaşan her isteğe basitçe güvenseydi, api-gateway'i atlayan HERHANGİ bir yol -- yanlış yapılandırılmış bir rota, bir iç ağda doğrudan ulaşılabilir bir servis, birinin gateway'den geçirmeyi unuttuğu gelecekteki bir servis -- hiçbir korumaya sahip olmazdı. Zero trust, order-service'in JWT'yi de KENDİSİ, bağımsız olarak doğrulaması anlamına gelir, "muhtemelen zaten kontrol edilmiştir" diye varsaymak yerine.

```
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

// order-service's OWN security configuration -- see "Why the Gateway Alone
// Isn't Enough: Zero Trust Between Services" for why order-service validates
// the SAME JWT api-gateway already validated, instead of trusting that a
// request reaching it must have already passed the gateway's check. This is
// the SERVLET-based config style (@EnableWebSecurity, HttpSecurity), matching
// order-service's blocking Spring MVC controllers -- unlike
// ApiGatewaySecurityConfig's reactive style, which matches api-gateway's
// WebFlux runtime.
@EnableWebSecurity
class OrderServiceSecurityConfig {

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .authorizeHttpRequests(requests -> requests
                        .requestMatchers("/actuator/health").permitAll()
                        // Creating an order requires the "customer" role --
                        // see "Authorization: Restricting an Endpoint by Role".
                        // Every OTHER authenticated request just needs a valid
                        // JWT, no specific role.
                        .requestMatchers(org.springframework.http.HttpMethod.POST, "
/orders")
                        .hasRole("customer")
                        .anyRequest().authenticated())
                .oauth2ResourceServer(oauth2 -> oauth2.jwt(jwt -> {}))
                .build();
    }
}
```

```
# Added to order-service's application.yml -- the SAME issuer-uri api-gateway
# uses, since both are verifying the SAME identity provider's tokens. This is
# what "zero trust between services" looks like in configuration: order-service
# doesn't ask api-gateway "did you already check this?" -- it verifies the
# token itself, independently, every time.

spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://auth.example.com/
```

> `OrderServiceJwtConfig.yml`'in `ApiGatewayJwtConfig.yml` ile AYNI `issuer-uri`'ye işaret ettiğine dikkat edin -- her iki servis de AYNI kimlik sağlayıcısından AYNI token'ları, tamamen bağımsız olarak doğruluyor. Hiçbir servis diğerine "bunu zaten kontrol ettin mi?" diye sormuyor.

Authorization: Bir Endpoint'i Role Göre Kısıtlamak

Bir istek authenticate olduktan sonra, `OrderServiceSecurityConfig`'in `.hasRole("customer")` kuralı (yukarıya bakınız) kimin özellikle sipariş verebileceğine dair AYRI kararı verir -- o role sahip olmayan authenticated bir kimlik, eksik ya da geçersiz bir token'ın üreteceği `401 Unauthorized` DEĞİL, `403 Forbidden` alır.

Kimliği Yaymak: Correlation Id'nin Security Karşılığı

order-service'in kendi JWT'si, `ResilientStockClient` üzerinden inventory-service'i çağırdığında (Resilience4j dersine bakınız) otomatik olarak yanında gitmez -- Observability dersinin correlation id için kapattığı AYNI boşluk, şimdi kimlik için.

```
import org.springframework.http.HttpRequest;
import org.springframework.http.client.ClientHttpRequestExecution;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.http.client.ClientHttpResponse;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.oauth2.jwt.Jwt;

import java.io.IOException;

// The security counterpart of the Observability lesson's
// RestClientCorrelationIdInterceptor -- same shape, different header. Without
// this, order-service's call to inventory-service (see ResilientStockClient in
// the Resilience4j lesson) would carry NO identity at all, and inventory-
// service's own SecurityFilterChain (built the same way as
// OrderServiceSecurityConfig) would have nothing to authenticate -- either the
// call fails outright, or inventory-service has to trust it unconditionally,
// which is exactly the gap "Why the Gateway Alone Isn't Enough" warns against
// one level further down the chain.
//
// Registered on the SAME @LoadBalanced RestClient.Builder bean the Service
// Discovery & Eureka lesson introduced -- adding this interceptor (alongside
// RestClientCorrelationIdInterceptor) doesn't change anything else about how
// the call is made, only which headers travel with it.
class RestClientBearerTokenInterceptor implements ClientHttpRequestInterceptor {

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body,
ClientHttpRequestExecution execution)
            throws IOException {
        // The Jwt Spring Security already validated for THIS incoming request
        // (see OrderServiceSecurityConfig) is available from the security
        // context -- forwarding its original, already-verified token value is
        // simpler and more honest than order-service minting a new one on
        // inventory-service's behalf.
        if (SecurityContextHolder.getContext().getAuthentication() instanceof
                org.springframework.security.oauth2.server.resource.authentication.
JwtAuthenticationToken jwtAuth) {
            Jwt jwt = jwtAuth.getToken();
            request.getHeaders().setBearerAuth(jwt.getTokenValue());
        }
        return execution.execute(request, body);
    }
}
```

Best Practices

- **İsteği alan HER serviste kimliği doğrula, yalnızca herkese açık internete bakanda değil** -- "Gateway Tek Başına Neden Yetmiyor" bölümüne bakınız.
- **Authentication ve authorization'ı ayrı konular olarak tut**, birbirine yakın yapılandırılmış olsalar bile (`OrderServiceSecurityConfig` 'te olduğu gibi) -- "Authentication vs Authorization" bölümüne bakınız.

- **Kimliği servis sınırları boyunca bilinçli olarak yay**, correlation id'nin yayıldığı AYNI şekilde -- `RestClientBearerTokenInterceptor`'a, ve onun ayna olduğu Observability dersinin `RestClientCorrelationIdInterceptor`'ına bakınız.
- **Health check endpoint'lerini herkese açık tut** (yukarıdaki `.pathMatchers("/actuator/health").permitAll()`'a bakınız) -- load balancer'ların ve orkestratörlerin bunlara bir token olmadan ulaşması gerekir.

Yaygın Hatalar

- **api-gateway'in authentication kontrolüne sistemin TEK güvenlik sınırı olarak güvenmek.** Başka bir yoldan ulaşılabilir herhangi bir iç servis, kimliği KENDİSİ doğrulamadıkça hiçbir korumaya sahip değildir -- "Gateway Tek Başına Neden Yetmiyor" bölümüne bakınız.
- **Bir `401`'i bir `403` ile karıştırmak.** `401 Unauthorized`, authentication'ın kendisinin başarısız olduğu anlamına gelir (token yok, ya da geçersiz); `403 Forbidden`, authentication'ın başarılı olduğu ama authorization'ın olmadığı anlamına gelir -- ikisini karıştırmak gerçek bir erişim sorununu debug etmeyi çok zorlaştırır.
- **Kimliği alt akıştaki bir servis çağrısına yaymayı unutmak.** `RestClientBearerTokenInterceptor` olmadan, inventory-service order-service'ten tamamen unauthenticated bir istek alır, ORİJİNAL dış istek düzgün authenticate edilmiş olsa bile.
- **Authorization mantığını bir controller metodunun içine dağınık `if` ifadeleri olarak koymak**, bir servisin diğer security kurallarının zaten yaşadığı yerde (`OrderServiceSecurityConfig`) tanımlamak yerine -- dağıtmak, bir servisin gerçek erişim kurallarını tek bir yerde denetlemeyi zorlaştırır.

Özet, Cheat Sheet ve Terimler Sözlüğü

Bir mikroservis sisteminde security, isteği alabilecek her servisin kimin sorduğuna (authentication) ve ne yapmasına izin olduğuna (authorization) kendi cevabına ihtiyaç duyması demektir -- yalnızca api-gateway'in kontrolüne güvenmek diğer her yolu korumasız bırakır. JWT'ler, yayınlayanın public key'ine sahip herhangi bir servisin bağımsız olarak kontrol edebileceği doğrulanabilir bir kimlik taşır -- bu, hem api-gateway'in hem order-service'in AYNI token'ı merkezi bir depoya geri çağrı yapmadan doğrulayabilmesini sağlayan şey. Kimlik, tıpkı bir correlation id gibi, servis sınırları boyunca bilinçli olarak yayılmalıdır.

Hızlı referans:

```
@EnableWebSecurity
class SomeServiceSecurityConfig {
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .authorizeHttpRequests(requests -> requests
                        .requestMatchers("/actuator/health").permitAll()
                        .anyRequest().authenticated())
                .oauth2ResourceServer(oauth2 -> oauth2.jwt(jwt -> {}))
                .build();
    }
}

// application.yml
// spring.security.oauth2.resourceserver.jwt.issuer-uri: https://auth.example.com/
```

Terimler Sözlüğü

Authentication — Bir isteği kimin yaptığını, tipik olarak bir token'ın imzasını kontrol ederek doğrulamak.

Authorization — Zaten authenticate olmuş bir kimliğin belirli bir şeyi yapmasına izin olup olmadığına karar vermek.

JWT (JSON Web Token) — Kendi kimlik claim'lerini taşıyan, yayınlayanın public key'ine sahip herhangi bir tarafça bağımsız olarak doğrulanabilir, kriptografik olarak imzalanmış kendi kendine yeterli bir token.

Zero Trust — Hiçbir servisin bir isteğin başka bir yerde zaten doğrulanmış olduğunu varsaymaması, kimliği kendisinin doğrulaması gerektiği ilkesi.

Resource Server — JWT'leri doğrulamak ve claim'lerine dayalı erişim kurallarını uygulamak üzere yapılandırılmış bir servis (burada order-service ya da api-gateway gibi).