1. Giriş (Introduction)

Geleneksel Kimlik Doğrulama Modeli ve Problemleri

- **Geleneksel Model:**
  İstemci, sunucudaki erişim kısıtlanmış bir kaynağa (protected resource) ulaşmak için kaynak sahibinin kimlik bilgilerini (örneğin, kullanıcı adı ve şifre) kullanır.
- **Üçüncü Taraf Uygulamalar:**
  Kaynak sahibi, üçüncü taraf bir uygulamaya erişim izni vermek istediğinde, kimlik bilgilerini (örneğin şifre) bu uygulama ile paylaşmak zorunda kalır.
  Ancak bu yaklaşım aşağıdaki sorunları doğurur:
  1. **Şifre Saklama Problemi:**
     Üçüncü taraf uygulamaların, kaynak sahibinin kimlik bilgilerini (genelde düz metin olarak) saklaması gerekir. Bu, güvenlik riski yaratır.
  2. **Şifre Doğrulama Zorunluluğu:**
     Sunucular, güvenlik açıkları olan şifre doğrulama mekanizmasını desteklemek zorunda kalır.
  3. **Gereksiz Geniş Yetki:**
     Üçüncü taraf uygulamalar, kaynak sahibinin tüm korunan kaynaklarına geniş erişim yetkisi alır. Kaynak sahibi, erişimi süre veya belirli kaynaklarla sınırlayamaz.
  4. **Erişim Kısıtlama Eksikliği:**
     Kaynak sahibi, belirli bir üçüncü taraf uygulamanın erişimini iptal edemez. Erişim kaldırmak için şifresini değiştirmek zorunda kalır, bu da tüm uygulamaların erişimini kaldırır.
  5. **Güvenlik Riski:**
     Üçüncü taraf uygulamalardan birinin güvenliği ihlal edilirse, kullanıcının şifresi ve şifre ile korunan tüm veriler tehlikeye girer.

OAuth ile Gelen Çözüm

OAuth, yukarıdaki sorunları çözmek için bir **yetkilendirme katmanı** (authorization layer) sunar ve istemciyi kaynak sahibinden ayırır:

1. **Erişim Token'ları:**
   Kaynak sahibinin kimlik bilgileri yerine, istemciye bir **erişim token'ı** (access token) verilir. Bu token:
   - Belirli bir erişim kapsamı (scope),
   - Belirli bir ömür süresi (lifetime),
   - Diğer erişim özelliklerini belirtir.
2. **Yetkilendirme Sunucusu:**
   - Erişim token'ları, kaynak sahibinin onayı ile bir **yetkilendirme sunucusu (authorization server)** tarafından oluşturulur.
   - İstemci, bu token'ı kullanarak kaynak sunucusundaki (resource server) korunan kaynaklara erişir.

Bir Örnek Senaryo

- **Senaryonun Tarafları:**
    - **Kaynak Sahibi (End-user):** Fotoğrafların sahibi.
    - **İstemci (Client):** Baskı hizmeti sağlayıcısı.
    - **Kaynak Sunucusu (Resource Server):** Fotoğraf paylaşım hizmeti.
    - **Yetkilendirme Sunucusu:** Fotoğraf paylaşım hizmetinin güvendiği bir kimlik doğrulama sunucusu.
- **Nasıl Çalışır:**
    - Kullanıcı, baskı hizmetine fotoğraflarına erişim izni verir.
    - Kullanıcı, doğrudan fotoğraf paylaşım hizmetinin güvendiği yetkilendirme sunucusunda kimlik doğrulaması yapar.
    - Yetkilendirme sunucusu, baskı hizmetine belirli bir erişim yetkisi veren bir erişim token'ı (delegation-specific access token) sağlar.
    - Baskı hizmeti, bu token'ı kullanarak kullanıcı fotoğraflarına erişir. Kullanıcının şifresi hiçbir zaman baskı hizmetine verilmez.

OAuth 2.0 ve HTTP

- **Protokol Uyumlu:**
  OAuth 2.0, HTTP protokolü (RFC 2616) ile uyumlu olacak şekilde tasarlanmıştır.
  HTTP dışında başka bir protokolde OAuth kullanımı bu dokümanın kapsamı dışındadır.

OAuth 1.0 ve OAuth 2.0

1. **Geçmiş:**
   OAuth 1.0 protokolü, küçük bir topluluğun çabalarıyla oluşturulmuş bir bilgi dokümanıdır (RFC 5849).
2. **Deneyim ve Gelişim:**
   OAuth 2.0, OAuth 1.0'ın uygulama deneyimlerinden ve daha geniş bir IETF topluluğunun ek gereksinimlerinden yola çıkarak geliştirilmiştir.
3. **Geriye Dönük Uyum (Backward Compatibility):**
    - OAuth 2.0, OAuth 1.0 ile **geriye dönük uyumlu değildir.**
    - İki protokol aynı ağda birlikte çalışabilir, ancak OAuth 2.0 yeni uygulamalar için tavsiye edilir.
    - OAuth 1.0 yalnızca mevcut uygulamaları desteklemek için kullanılmalıdır.
4. **Farklı Yapılar:**
   OAuth 2.0, OAuth 1.0 ile çok az uygulama detayı paylaşır. OAuth 1.0'ı bilenlerin, bu dokümanı **ön yargısız** bir şekilde incelemeleri önerilir.

Öne Çıkan Detaylar

1. OAuth 2.0, istemcinin kaynak sahibinin kimlik bilgilerine erişmeden, güvenli ve sınırlı bir şekilde kaynaklara erişmesini sağlar.
2. Yetkilendirme ve erişim token’ları aracılığıyla erişim daha güvenli, esnek ve sınırlı hale getirilmiştir.
3. OAuth 2.0, önceki versiyonun eksiklerini kapatmak ve modern ihtiyaçlara cevap vermek için yeniden tasarlanmıştır.

1.1. Roller (Roles)

OAuth protokolünde dört ana rol tanımlanmıştır:

---

1. Resource Owner (Kaynak Sahibi)

- **Tanım:**
  Korunan bir kaynağa erişim izni verebilen varlık. Bu genellikle bir kişidir, bu durumda **end-user** (son kullanıcı) olarak adlandırılır.
- **Örnek Senaryo:**
  Bir kullanıcı (örneğin, Alice), sosyal medya platformundaki fotoğraflarına erişim izni verebilir. Burada kullanıcı, kaynak sahibidir.

---

2. Resource Server (Kaynak Sunucusu)

- **Tanım:**
  Korunan kaynakları barındıran ve erişim token'larını kullanarak gelen istekleri kabul edip yanıtlayan sunucu.
- **Görevi:**
  - Erişim token'larını doğrular.
  - Korunan kaynaklara erişim sağlar.
- **Örnek Senaryo:**
  Alice'in fotoğraflarını barındıran sosyal medya platformunun sunucusu, bir **resource server**dır. Bu sunucu, istemciden gelen erişim token'ını kontrol eder ve token geçerliyse fotoğraflara erişim izni verir.

---

3. Client (İstemci)

- **Tanım:**
  Kaynak sahibinin adına korunan kaynaklara erişim isteği gönderen uygulama. İstemci, kaynak sahibinin izni ve yetkilendirmesiyle çalışır.
  Not: İstemci, belirli bir teknolojiye veya platforma bağlı değildir. Örneğin, bir web uygulaması, masaüstü yazılımı veya mobil uygulama olabilir.
- **Görevi:**
  - Kaynak sahibinden izin alarak bir **access token** talep eder.
  - Bu token'ı kullanarak korunan kaynaklara erişim talebinde bulunur.
- **Örnek Senaryo:**
  Alice, bir fotoğraf baskı hizmeti (örneğin, Printify) kullanır. Printify, Alice'in sosyal medya platformundaki fotoğraflarına erişmek için istemci olarak davranır.

---

4. Authorization Server (Yetkilendirme Sunucusu)

- **Tanım:**
  Kaynak sahibini başarıyla kimlik doğruladıktan ve gerekli izinleri aldıktan sonra istemciye erişim token'ları sağlayan sunucu.
- **Görevi:**
  - Kaynak sahibinin kimlik doğrulamasını yapar (örneğin, kullanıcı adı ve şifre ile).
  - Kaynak sahibinden istemciye yetki verir.
  - Erişim token'ı (access token) oluşturur ve istemciye iletir.
- **Örnek Senaryo:**
  Alice, sosyal medya platformuna giriş yaparak fotoğraf baskı hizmetine erişim izni verir. Bu süreçte sosyal medya platformunun yetkilendirme sunucusu devreye girer ve gerekli token'ları üretir.

Yetkilendirme Sunucusu ve Kaynak Sunucusu Etkileşimi

- **Kapsam Dışı:**
  Yetkilendirme sunucusu ve kaynak sunucusu arasındaki etkileşim bu spesifikasyonun kapsamı dışındadır. Bu iki rol, aynı sunucuda birleştirilebilir veya farklı sunuculara dağıtılabilir.
- **Paylaşılabilir Yetkilendirme Sunucusu:**
  - Tek bir yetkilendirme sunucusu, birden fazla kaynak sunucusuna erişim token'ı sağlayabilir.
  - Örneğin, bir yetkilendirme sunucusu hem bir sosyal medya platformuna hem de bir dosya paylaşım platformuna erişim için token'lar üretebilir.

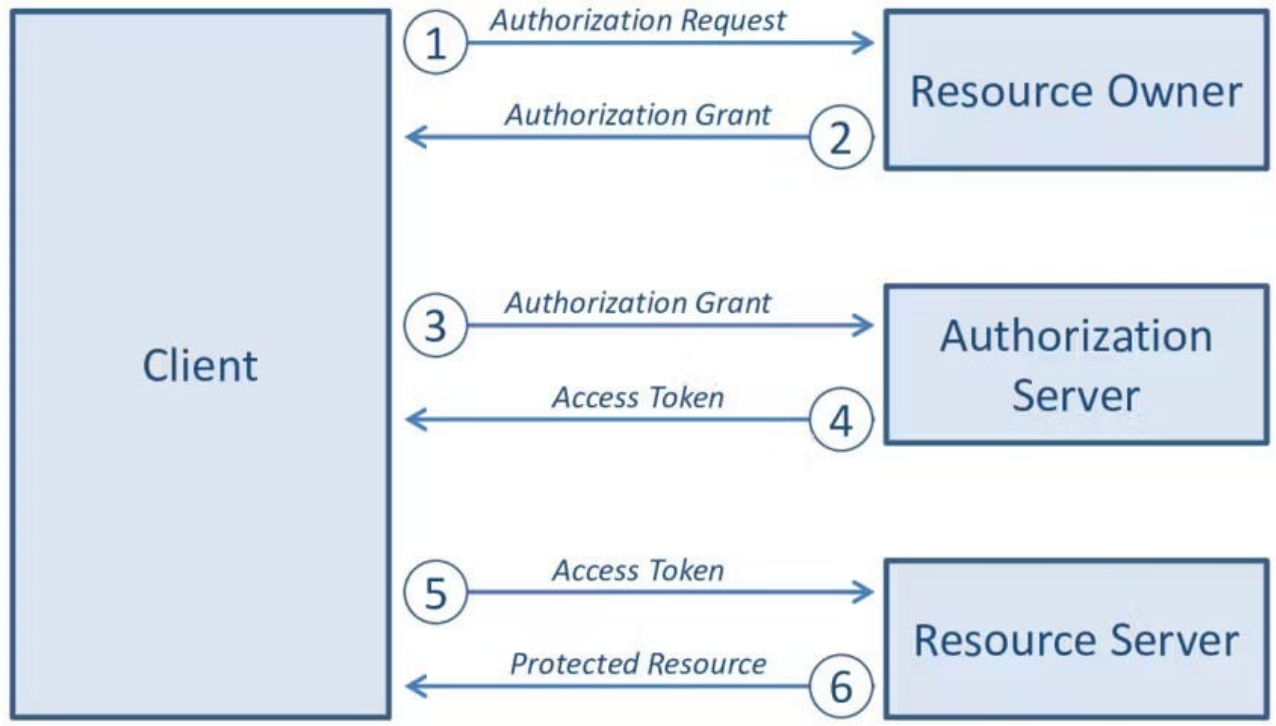
Özet

OAuth’un dört ana rolü şunları içerir:

1. **Resource Owner (Kaynak Sahibi):** Korunan kaynakların sahibidir ve izin verebilir.
2. **Resource Server (Kaynak Sunucusu):** Korunan kaynaklara erişimi kontrol eder.
3. **Client (İstemci):** Kaynak sahibinin adına hareket ederek kaynaklara erişim ister.
4. **Authorization Server (Yetkilendirme Sunucusu):** Kimlik doğrulama ve yetkilendirme işlemlerini gerçekleştirir, erişim token'ları oluşturur.

Şema (Abstract Protocol Flow)

Şemada dört ana rolün (Client, Resource Owner, Authorization Server, Resource Server) nasıl etkileşimde bulunduğu ve OAuth 2.0 sürecinin adım adım nasıl ilerlediği gösterilmiştir:



Bu şema, protokolün akışını altı adımda açıklıyor. Her bir adımı detaylı olarak ele alalım:

Adımlar

(1) Authorization Request (Yetkilendirme İsteği)

- **Ne Olur?**
    - **Client (İstemci)**, **Resource Owner (Kaynak Sahibi)**'nden yetkilendirme ister.
    - Bu istek, doğrudan kaynak sahibine yapılabilir, ancak genellikle **Authorization Server (Yetkilendirme Sunucusu)** bir aracı olarak kullanılır.
- **Örnek:**
    - Bir kullanıcı, bir mobil uygulamanın sosyal medya hesaplarına erişmesine izin vermek için giriş yapar.

---

(2) Authorization Grant (Yetkilendirme Belgesi)

- **Ne Olur?**
    - **Resource Owner**, istemciye bir "authorization grant" (yetkilendirme belgesi) verir.
    - Bu belge, istemcinin korunan kaynaklara erişim iznini temsil eder.
    - Bu belge dört türden biri olabilir:

        - Authorization Code
        - Implicit Grant
        - Resource Owner Password Credentials
        - Client Credentials
- **Örnek:**
    - Kullanıcı, sosyal medya platformu üzerinden uygulamaya erişim izni verdiğinde, uygulama bir "authorization code" alır.

(3) Access Token Request (Erişim Token'ı İsteği)

- **Ne Olur?**
    - **Client**, **Authorization Server** ile iletişime geçer.
    - Daha önce aldığı "authorization grant" belgesini sunarak bir "access token" talep eder.
- **Örnek:**
    - Mobil uygulama, sosyal medya platformuna, aldığı "authorization code" ile birlikte bir "access token" talebi gönderir.

---

(4) Access Token Issuance (Erişim Token'ının Verilmesi)

- **Ne Olur?**
    - **Authorization Server**, istemciyi ve "authorization grant" belgesini doğrular.
    - Her şey uygunsa istemciye bir "access token" sağlar.
- **Örnek:**
    - Sosyal medya platformu, doğrulama işlemlerini başarıyla tamamladıktan sonra uygulamaya bir "access token" verir.

---

(5) Protected Resource Request (Korunan Kaynak İsteği)

- **Ne Olur?**
    - **Client**, **Resource Server**'a erişmek istediği korunan kaynak için bir istek gönderir.
    - Bu isteği "access token" ile birlikte sunar.
- **Örnek:**
    - Mobil uygulama, kullanıcıya ait fotoğrafları almak için sosyal medya platformunun API'sine "access token" ile bir istek gönderir.

---

(6) Resource Access (Kaynağa Erişim)

- **Ne Olur?**
    - **Resource Server**, sunulan "access token"ı doğrular.
    - Token geçerliyse korunan kaynağı istemciye sağlar.

- **Örnek:**
    - Sosyal medya platformu, "access token"ın geçerli olduğunu doğrular ve kullanıcının fotoğraflarını uygulamaya iletir.

Notlar

1. **Authorization Server'ın Aracı Rolü:**
    - **(1)** ve **(2)** adımlarında, **Authorization Server** genellikle aracı olarak kullanılır.
    - Bu, kullanıcı deneyimini kolaylaştırır ve güvenliği artırır.
2. **Access Token:**
    - "Access token", istemcinin kaynak sunucusuna erişim için kullanacağı geçici bir kimlik belgesidir.
    - Token, belirli bir kapsam (scope) ve süreyle sınırlıdır.
3. **Bağımsız Roller:**
    - **Authorization Server** ve **Resource Server**, aynı sunucuda çalışabilir veya farklı sunucular olarak yapılandırılabilir.
    - Tek bir yetkilendirme sunucusu, birden fazla kaynak sunucusuna erişim sağlayabilir.

1.3 - Authorization Grant

OAuth 2.0 protokolünde, istemcinin (client) bir **Access Token** almak için kullandığı kimlik belgesidir. Bu belge, **Resource Owner**'ın (kaynak sahibinin) korunan kaynaklara erişim iznini temsil eder.

Bu bağlamda, OAuth 2.0 dört temel **Authorization Grant** türünü tanımlar:

1. **Authorization Code Grant**: Güvenli ve iki adımlı bir süreçtir. İstemci, önce bir "authorization code" alır, ardından bu kodu kullanarak **Access Token** talep eder.
2. **Implicit Grant**: Daha hızlıdır ve genellikle tarayıcı tabanlı uygulamalarda kullanılır. Ancak güvenlik açısından daha az güçlüdür, çünkü **Access Token** doğrudan istemciye sunulur.
3. **Resource Owner Password Credentials Grant**: İstemcinin, kullanıcının kullanıcı adı ve şifresini doğrudan **Authorization Server**'a gönderdiği bir yöntemdir. Güvenilir istemciler için uygundur, ancak genellikle önerilmez.
4. **Client Credentials Grant**: İstemci, kendine ait kimlik bilgilerini kullanarak **Authorization Server**'dan bir **Access Token** talep eder. Genellikle kullanıcıya bağlı olmayan istemci-istemci iletişimleri için kullanılır.

Ayrıca, OAuth 2.0, özel gereksinimler için yeni **Grant** türlerinin tanımlanabilmesine olanak tanıyan bir genişletilebilirlik mekanizması sunar.

1.3.1 - Authorization Code

OAuth 2.0 protokolünde, istemcinin **Access Token** alması için kullanılan bir yetkilendirme türüdür ve en güvenli yöntemlerden biri olarak kabul edilir. Bu mekanizma, istemci ile kaynak sahibi (resource owner) arasında doğrudan bir bağlantı kurmak yerine, bir **Authorization Server**'ın aracı olarak kullanılması prensibine dayanır. İşleyişi şu şekildedir:

1. **Authorization Server Aracılığıyla Yetkilendirme**:
    - İstemci, kaynak sahibinden doğrudan yetkilendirme istemek yerine, kaynak sahibini **Authorization Server**'a yönlendirir.
    - Bu yönlendirme, genellikle kaynak sahibinin kullanıcı arayüzü veya tarayıcı (user-agent) aracılığıyla yapılır.
2. **Kaynak Sahibinin Kimlik Doğrulaması ve Yetkilendirilmesi**:
    - **Authorization Server**, kaynak sahibinin kimlik bilgilerini doğrular ve erişim yetkisini onaylar.
    - Bu süreçte kaynak sahibinin kullanıcı adı veya şifresi gibi kimlik bilgileri, istemciyle paylaşılmaz. Bu, istemciyle hassas bilgiler arasındaki doğrudan temas riskini ortadan kaldırır.
3. **Authorization Code'un İstemciye Dönmesi**:
    - **Authorization Server**, kaynak sahibini istemciye yönlendirirken bir **Authorization Code** sağlar.
    - Bu kod, istemcinin daha sonra bir **Access Token** talep etmek için kullanacağı bir geçici kimlik belgesidir.
4. **Access Token Talebi**:

- İstemci, aldığı **Authorization Code** ile **Authorization Server**'a bir **Access Token** talebinde bulunur.
- Bu adımda istemci kendini **Authorization Server**'a kimlik bilgileriyle doğrulayarak ek bir güvenlik katmanı sağlar.

Güvenlik Faydaları:

- **Kimlik Doğrulama Güvencesi**: **Authorization Server**, istemcinin kimliğini doğrular.
- **Access Token'ın Doğrudan İletilmesi**: **Access Token**, istemciye doğrudan iletilir ve kaynak sahibinin tarayıcısı üzerinden geçmediği için maruz kalma riski azalır.
- **Kaynak Sahibinin Gizliliği**: Kaynak sahibinin kimlik bilgileri (örneğin, kullanıcı adı ve şifre) istemciyle asla paylaşılmaz, yalnızca **Authorization Server** ile paylaşılır.

Bu yöntem, güvenliğin ve hassas bilgilerin korunmasının ön planda olduğu istemci-tarayıcı etkileşimlerinde yaygın olarak tercih edilir.

1.3.2 - Implicit grant

Tarayıcı tabanlı uygulamalar (örneğin JavaScript ile yazılmış istemciler) için optimize edilmiş ve OAuth 2.0 sürecini basitleştiren bir yetkilendirme türüdür. Bu tür, daha az işlem gerektirdiği için **Authorization Code** yöntemine kıyasla daha hızlıdır, ancak bazı güvenlik zafiyetleri içerir. İşleyişi ve özellikleri şu şekilde açıklanabilir:

İşleyiş:

1. **Erişim Tokenı Doğrudan Verilir**:
    - **Authorization Server**, istemciye bir **Authorization Code** yerine doğrudan bir **Access Token** sağlar.
    - **Access Token**, kaynak sahibinin yetkilendirme süreci tamamlandıktan hemen sonra istemciye iletilir.
    - Arada bir **Authorization Code** alma ve bu kodu kullanarak token isteme adımları atlanır.
2. **İstemci Kimlik Doğrulaması Olmaz**:
    - Bu akışta **Authorization Server**, istemciyi kimlik doğrulamasından geçirmez.
    - Bunun yerine, istemcinin kimliğini doğrulamak için genellikle istemcinin kullandığı **Redirect URI** gibi faktörlere güvenilir.
3. **Tarayıcı Üzerinden İşleme**:
    - Token, kaynak sahibinin tarayıcısı (user-agent) üzerinden istemciye iletilir.
    - Bu, tokenın tarayıcıya veya başka bir uygulamaya maruz kalma riskini artırır.

Avantajlar:

- **Hız ve Verimlilik**:

    - İstemcinin **Access Token** alması için gereken işlem sayısı azaltıldığı için daha hızlıdır.
    - Özellikle tek sayfa uygulamalar (SPA) gibi tarayıcı tabanlı istemcilerde performansı artırır.
- **Basitleştirilmiş Akış**:
    - Kullanıcı deneyimini iyileştiren ve daha az adım gerektiren bir yapı sağlar.

Güvenlik Riskleri:

1. **Tokenın Maruz Kalma Riski**:
    - **Access Token**, kaynak sahibinin tarayıcısı üzerinden istemciye iletildiğinden, tarayıcıya veya başka bir uygulamaya maruz kalabilir.
    - Eğer tarayıcıda açıklar veya kötü amaçlı yazılımlar varsa, tokenın ele geçirilme riski yüksektir.
2. **İstemcinin Doğrulanmaması**:
    - İstemci kimlik doğrulaması yapılmadığı için kötü niyetli istemcilerin **Access Token** alması kolaylaşabilir.
3. **Token Süresinin Kısa Olması Gerekliliği**:
    - Bu akışta güvenlik risklerini en aza indirmek için token süreleri çok kısa tutulmalıdır.

Nerelerde Kullanılır?

- **Tek Sayfa Uygulamalar (Single Page Applications - SPA)**:
    - React, Angular gibi tarayıcı tabanlı çerçevelerle yazılmış uygulamalarda sıkça tercih edilir.
- **Sunucusuz (Serverless) Uygulamalar**:
    - İstemcinin yalnızca tarayıcıda çalıştığı ve backend sunucusunun olmadığı durumlarda uygundur.

Güvenlik Açısından Değerlendirme:

Authorization Code Grant, güvenlik açısından daha güçlüdür ve mümkünse tercih edilmelidir. Ancak, yalnızca tarayıcı tabanlı bir uygulama kullanılıyorsa ve hızlı bir akış gerekiyorsa, **Implicit Grant** seçeneği kullanılabilir. Kullanım sırasında, güvenlik risklerini azaltmak için şunlara dikkat edilmelidir:

- Token süresi kısa tutulmalı.
- **Redirect URI** kesinlikle doğrulanmalı.
- Tarayıcı güvenliğinin sağlandığından emin olunmalı.

Sonuç olarak, **Implicit Grant**, belirli durumlarda performans avantajı sağlarken güvenlik açıkları barındırdığı için dikkatli bir şekilde değerlendirilmesi gereken bir yöntemdir.

1.3.3 - Resource Owner Password Credentials (ROPC)

Kaynak sahibinin kullanıcı adı ve şifre bilgilerini doğrudan bir yetkilendirme belgesi (grant) olarak kullanarak istemciye erişim tokenı sağlama yöntemidir. Bu yöntem yalnızca kaynak sahibi ile istemci arasında yüksek güven ilişkisi olduğu durumlarda kullanılmalıdır. Örneğin, istemci işletim sisteminin bir parçası olduğunda veya çok yetkili bir uygulama olduğunda tercih edilebilir. İşleyişi ve özellikleri şu şekilde özetlenebilir:

İşleyiş:

1. **Kullanıcı Adı ve Şifre Kullanımı**:
    - Kaynak sahibi (örneğin bir kullanıcı), istemciye kullanıcı adı ve şifresini doğrudan verir.
    - İstemci, bu kimlik bilgilerini **Authorization Server**'a gönderir.
2. **Erişim Tokenı Sağlanması**:
    - **Authorization Server**, bu kimlik bilgilerini doğrular.
    - Eğer doğrulama başarılı olursa, istemciye bir **Access Token** sağlar.
    - Token, kaynak sahibinin korunan kaynaklarına erişim için kullanılır.
3. **Kimlik Bilgilerinin Tek Seferlik Kullanımı**:
    - Kaynak sahibinin kullanıcı adı ve şifre bilgileri, yalnızca token almak için kullanılır ve başka bir amaçla saklanmaz.
    - Uzun süreli bir **Access Token** veya **Refresh Token** ile kimlik bilgilerini saklama ihtiyacı ortadan kalkar.

Avantajlar:

- **Basitlik**:
    - Diğer grant türleri (örneğin Authorization Code veya Implicit) yerine daha doğrudan ve basit bir yöntem sunar.
- **Kimlik Bilgilerini Saklama Zorunluluğunun Azalması**:
    - İstemci, kullanıcı adı ve şifre gibi hassas bilgileri uzun süreli saklamak yerine bunları bir token ile değiştirir.
- **Yetkilendirme Türünün Alternatif Olmadığı Durumlar İçin Uygun**:
    - Örneğin, bir cihazın işletim sistemi parçası olan istemcilerde ya da başka grant türlerinin uygulanamayacağı durumlarda kullanışlıdır.

Güvenlik Riskleri ve Dikkat Edilmesi Gerekenler:

1. **Kimlik Bilgilerinin Ele Geçirilme Riski**:
    - Kullanıcı adı ve şifre bilgilerinin istemciye doğrudan verilmesi, büyük bir güvenlik riski doğurabilir.

    - İstemcinin güvenilir ve güvenli bir şekilde çalıştığından emin olunmalıdır.
2. **Sınırlı Kullanım Alanı**:
    - Yalnızca istemci ile kaynak sahibi arasında yüksek güven ilişkisi olduğu durumlarda kullanılmalıdır.
    - Düşük güven seviyesine sahip istemciler için uygun değildir.
3. **Alternatif Grant Türlerinin Tercihi**:
    - Mümkünse **Authorization Code Grant** gibi daha güvenli grant türleri kullanılmalıdır.
    - ROPC, yalnızca başka bir seçeneğin olmadığı durumlarda bir "son çare" olarak değerlendirilmelidir.

Nerelerde Kullanılır?

- **Cihaz Tabanlı Uygulamalar**:
    - Örneğin, işletim sistemi tarafından entegre edilmiş bir istemci uygulamasında kullanılabilir.
- **Yüksek Güven Gerektiren Uygulamalar**:
    - Uygulamanın doğrudan kontrol edilen bir ortamda çalıştığı ve güvenliğin tamamen sağlandığı senaryolarda uygundur.

Değerlendirme ve Sonuç:

Resource Owner Password Credentials Grant, güvenliğin ve gizliliğin kritik olduğu durumlarda dikkatli bir şekilde kullanılmalıdır. Bu grant türü, istemcinin kaynak sahibinin kullanıcı adı ve şifre bilgilerine doğrudan erişim gerektirdiği için, kötüye kullanım riskini beraberinde getirir. Bu nedenle:

- Uygulama, yalnızca yüksek güven ortamlarında çalıştırılmalı.
- Kullanıcı adı ve şifre bilgileri yalnızca bir kere kullanılmalı ve hiçbir şekilde saklanmamalı.
- Mümkünse, daha güvenli grant türleri tercih edilmelidir.

Bu yöntem, yalnızca belirli ve sınırlı kullanım senaryolarında, güvenilir istemcilerle uygulanmalıdır.

1.3.4 - Client Credentials

İstemcinin kimliğini doğrulamak ve kaynaklara erişim izni almak için kullanılan bir yetkilendirme türüdür. Bu grant türü genellikle **istemcinin kendi adına** veya **önceden yetkilendirilmiş erişim** ile korunan kaynaklara erişim sağlamak amacıyla kullanılır. Başka bir deyişle, istemci, genellikle kaynak sahibiyle aynı kimliklere sahip değildir ve yalnızca **kendi kaynaklarına** erişim talep eder.

İşleyiş:

1. **Kimlik Doğrulama**:

    - İstemci, bir **access token** almak için **Authorization Server**'a kimlik bilgilerini (client ID ve client secret gibi) sunar.
    - Bu kimlik bilgileri istemcinin güvenli bir şekilde doğrulanmasını sağlar.
2. **Erişim Tokenı Alınması**:
    - İstemci, **client credentials**'ını kullanarak **authorization server** üzerinden **access token** talep eder.
    - Bu işlem, istemcinin kaynak sahibinin kimlik bilgilerine veya şifresine ihtiyacı olmadığı için daha güvenli ve hızlıdır.
3. **Erişim Sağlanması**:
    - Erişim tokenı alındıktan sonra istemci, bu token'ı kullanarak **resource server**'a (korunan kaynak sunucusu) erişebilir.

Ne Zaman Kullanılır?

- **İstemci Kendi Kaynaklarına Erişmek İstediğinde**:
    - Eğer istemci, sadece kendi kontrolündeki kaynaklara erişim talep ediyorsa (örneğin, kendi veritabanındaki verilere), client credentials grant türü uygundur.
- **Önceden Yetkilendirilmiş Erişim Senaryoları**:
    - Eğer bir istemci, belirli bir kaynak üzerinde önceden anlaşmaya varmışsa (örneğin, bir API üzerinden belirli verilere erişim için önceden yetkilendirilmişse), bu tür bir grant kullanılabilir.
- **Sunucu-Sunucu İletişimi**:
    - Client credentials grant türü, genellikle **sunucu-sunucu iletişimi** için kullanılır. Örneğin, bir microservice mimarisinde, mikroservislerin kendi aralarında güvenli bir şekilde iletişim kurması gerektiğinde bu tür bir grant kullanılır.

Avantajlar:

1. **Güvenli ve Basit**:
    - Kaynak sahibinin kimlik bilgilerini istemciyle paylaşma gereksinimi yoktur. İstemci yalnızca kendi kimlik bilgilerini (client ID ve client secret) kullanarak kimlik doğrulaması yapar.
2. **İstemcinin Kendi Kaynaklarıyla İşlem Yapması**:
    - İstemci, yalnızca kendi kaynaklarına veya önceden yetkilendirilmiş kaynaklara erişim talep ettiği için, diğer kullanıcıların kaynaklarıyla ilgili bir risk oluşmaz.
3. **Sunucu-Sunucu Erişimi İçin Uygun**:
    - Genellikle API'ler veya mikro servisler gibi sunucu tabanlı uygulamalar arasında kullanılır. Bu tür uygulamalar genellikle kullanıcı adı ve şifre gerektirmeden yalnızca istemci kimlik bilgileriyle kimlik doğrulaması yapar.

Sınırlamalar ve Güvenlik Riskleri:

1. **Kaynak Sahipliği**:
    - Bu grant türü, **istemcinin kaynak sahibiyle aynı olması** durumunda kullanılabilir. Yani istemci, yalnızca kendi kaynakları üzerinde işlem yapabilir. Başka bir deyişle, istemcinin, kaynak sahibi olmayan veriler üzerinde işlem yapması için başka bir grant türü gereklidir.
2. **Client Secret'ın Güvenliği**:
    - **Client Secret** gibi kimlik bilgileri, istemcinin güvenli bir ortamda tutulması gerektiği için doğru güvenlik önlemleri alınmalıdır. Eğer bu bilgiler ele geçirilirse, kötü niyetli kişiler istemci adına yetkisiz erişimler yapabilir.
3. **Kısıtlı Erişim**:
    - Bu grant türü, yalnızca istemcinin önceden belirlenmiş ve yetkilendirilmiş kaynaklarına erişim sağlar. Diğer kaynaklara erişim için farklı bir grant türü gereklidir.

Sonuç:

Client Credentials Grant, özellikle istemcinin kendi kontrolündeki kaynaklara erişim sağlamak için tasarlanmış güvenli bir yetkilendirme türüdür. Bu yöntem, istemcilerin kimlik doğrulaması yapmasını sağlar, ancak kaynak sahibiyle aynı kimlikleri taşımayan istemciler için bu yöntem dışında başka grant türleri kullanılmalıdır.

Bu grant türü, genellikle **sunucu-sunucu iletişimi** ve **API entegrasyonları** için uygun olup, yüksek güvenlik önlemleri gerektiren ortamlarda uygulanmalıdır.

Access Token (Erişim Token'ı)

Access token, korunan kaynaklara erişim sağlamak için kullanılan bir kimlik doğrulama aracıdır. Bu token, istemcinin kaynaklara erişim hakkını temsil eden bir dizedir ve genellikle istemci tarafından şeffaf bir biçimde kullanılır. Access token, kaynak sahibi tarafından istemciye verilen yetkilere dayalı olarak belirli **kapsamlar (scope)** ve **erişim süreleri** içerir. Bu bilgiler, **resource server (kaynak sunucu)** ve **authorization server (yetkilendirme sunucu)** tarafından denetlenir.

Erişim Token'ının Temel Özellikleri:

1. **Kimlik Doğrulama Aracı:**
    - Access token, bir istemcinin kaynaklara erişim hakkını temsil eder. Bu, istemcinin, belirli bir **kapsama** ve **süreye** sahip olarak kaynaklara erişmesini sağlar.
2. **Opaklık (Opaque) Yapı:**
    - Token, genellikle istemci için **opak** yani şeffaf olmayan bir yapıdadır. Bu, istemcinin token'ın içeriğini anlamadığı, ancak bu token'ın geçerliliğini **resource server**'a sunarak erişim alabileceği anlamına gelir.
    - Bununla birlikte, bazı token'lar **kendisi içinde kimlik doğrulama bilgilerini** barındırabilir ve doğrulama yapıldığında bu bilgiler **verifiye** edilebilir.
3. **Kapsamlar ve Süreler:**
    - Erişim token'ları, **kaynak sahibi tarafından verilen yetkiler** doğrultusunda **belirli bir erişim kapsamı** ve **geçerlilik süresi** içerir. Örneğin, bir token yalnızca fotoğraf yüklemek için geçerli olabilir, başka bir token ise fotoğraf okuma yetkisi verebilir.
    - Token'ın geçerlilik süresi sona erdiğinde, istemcinin yeniden yetkilendirilmesi veya yenilenen bir token alması gerekebilir.
4. **Token Formatı:**
    - **Access token**'ların formatları farklılık gösterebilir. Bazı token'lar yalnızca bir kimlik (identifier) taşıyabilir, bazen ise **veri ve imza** içerebilir. Bu tür token'lar genellikle **JWT (JSON Web Token)** gibi standartlarla ifade edilir ve içerikleri doğrulanabilir. Örneğin, bir JWT token'ı, verileri ve bir dijital imzayı içerebilir, bu da onu doğrulamak için kullanılan **public key**'in gerekliliğini ortaya koyar.
5. **Gizlilik ve Güvenlik:**
    - **Access token**'lar genellikle **gizlidir**. Token'ın içeriği, yalnızca ilgili **resource server** tarafından anlaşılır ve doğrulanabilir. Bu, istemcinin token'ı doğru şekilde kullanabilmesi için ek kimlik doğrulama bilgilerine sahip olabileceği anlamına gelir. Ancak bu bilgiler, bu spesifikasyonun kapsamı dışında kalmaktadır.
6. **Tekrar Kullanılabilirlik:**
    - **Access token** tek bir kullanım için verilir, yani istemci bu token'ı kullanarak korunan kaynağa bir erişim isteği gönderir ve **resource server**, token'ı doğrulayıp yetkilendirme kararını verir.

Erişim Token'ının Avantajları:

1. **Basitleştirilmiş Kimlik Doğrulama:**
    - **Erişim token'ları**, farklı kimlik doğrulama yöntemlerini (örneğin, kullanıcı adı ve şifre gibi) tek bir token ile değiştirebilir. Bu, **resource server**'ın, farklı kimlik doğrulama yöntemlerini anlamak zorunda olmadan, **token**'ı doğrulayıp erişimi yönetmesine olanak tanır.
2. **Kapsamlı Güvenlik Sağlar:**
    - **Access token**'lar, yalnızca belirli bir **kapsama** ve **süreye** dayanarak sınırlı erişim sağlar. Bu, istemcinin yalnızca yetkilendirildiği kaynaklara ve işlevlere erişmesini sağlar.
3. **Esneklik ve Çeşitli Kullanım Yöntemleri:**
    - Token'lar farklı **formatlar** ve **kriptografik yöntemler** kullanılarak yapılandırılabilir. Bu, **resource server**'ın güvenlik gereksinimlerine göre token'ların biçimini ve kullanımını esnek bir şekilde uyarlamasına olanak tanır.
4. **Kapsamlı Uygulama Alanı:**
    - Bu token, çeşitli **kaynak sunucuları** tarafından kabul edilebilir. Yani bir istemci, birden fazla kaynağa erişim talep edebilir ve her bir kaynak için geçerli olan bir token alabilir.

Token'ın Yapısı ve Kullanımı:

1. **Şeffaf Olmayan Token (Opaque Token):**
    - Bu tür token, istemciye ne içerdiği hakkında bilgi vermez. Token'ı yalnızca **resource server** doğrulayabilir. Bu tür token'lar genellikle güvenlik amacıyla tercih edilir.
2. **JSON Web Token (JWT) gibi Yapılar:**
    - Bazı durumlarda, token'lar **JWT** gibi standartlara dayanır. JWT, hem verileri hem de bir imzayı içerir ve bu imza sayesinde token doğrulama yapılabilir. Bu tür token'lar **self-contained** yani kendi kendine doğrulanabilir yapıdadır.

Özet:

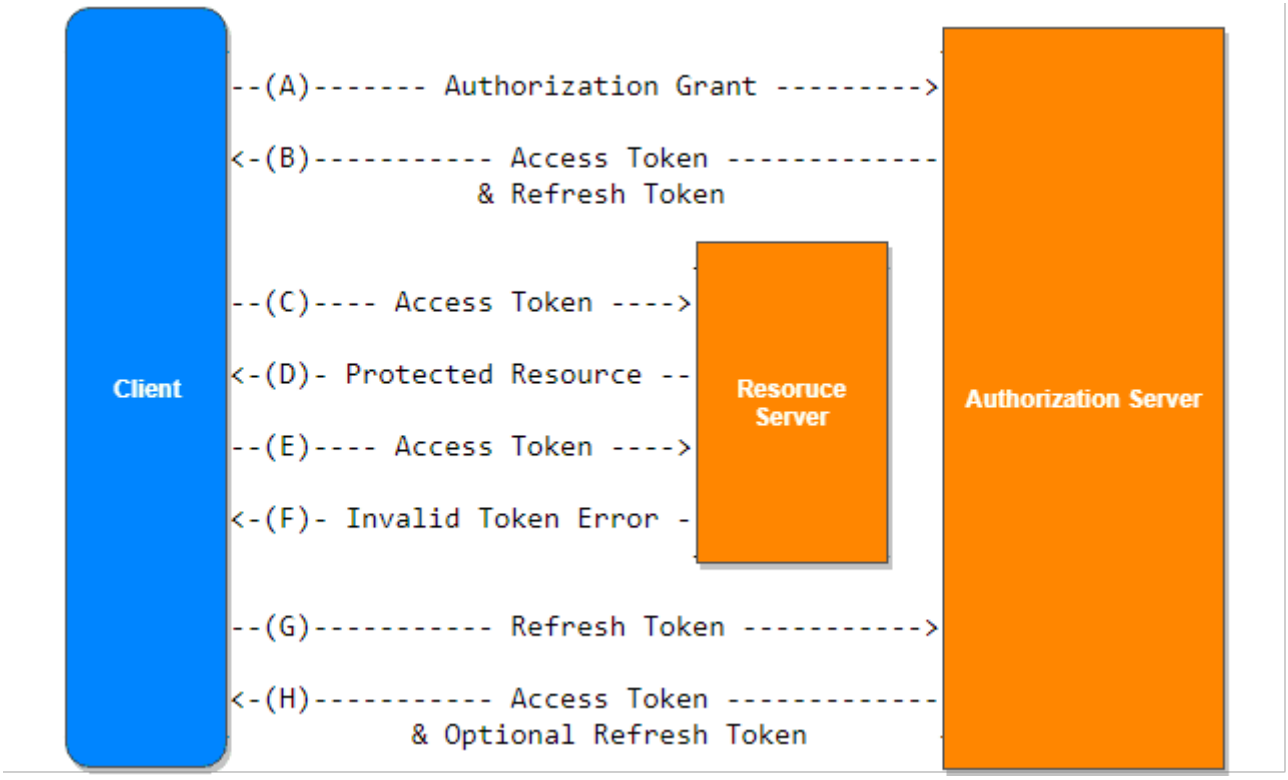
Access token, OAuth 2.0 protokolünde istemcilerin, kaynaklara erişim izni almasını sağlayan bir kimlik doğrulama aracıdır. Token, erişim hakkı tanımlayan ve sınırlayan bir dizedir ve istemci tarafından kullanılarak korunan kaynaklara erişim sağlanır. Token'ın yapısı ve güvenlik seviyesi, **resource server**'ın güvenlik gereksinimlerine göre farklılık gösterebilir. Ancak temel amacı, kaynaklara erişimi yönetmek ve kimlik doğrulama işlemini daha verimli hale getirmektir.

Refresh Token (Yenileme Token'ı)

Refresh token (yenileme token'ı), **access token**'ın süresi dolduğunda veya geçersiz hale geldiğinde yeni bir **access token** almak için kullanılan bir kimlik doğrulama aracıdır. Refresh token, istemciye **authorization server (yetkilendirme sunucusu)** tarafından verilir ve **resource server (kaynak sunucu)** ile doğrudan kullanılmaz. Refresh token, istemcinin sürekli olarak erişim sağlamak için yeni **access token**'lar alabilmesine imkan tanır.

Refresh Token'ın Temel Özellikleri:

1. **Yeni Access Token Alma:**
    - Refresh token, geçerli bir **access token**'ın süresi dolduğunda veya token geçersiz hale geldiğinde yeni bir access token almak için kullanılır. Access token'ın süresi genellikle kısa olur, bu yüzden refresh token ile yeni token almak gerekir.
2. **Sadece Yetkilendirme Sunucusunda Kullanım:**
    - Refresh token yalnızca **authorization server** ile etkileşime girmekte kullanılabilir. **Resource server** (kaynak sunucu) refresh token'ı almaz ve bu token herhangi bir şekilde bu sunuculara iletilmez.
3. **Genellikle Opak Yapı:**
    - Refresh token genellikle istemci için **opak** (şeffaf olmayan) bir dizedir. İstemci, token'ın içeriğini doğrudan göremez; ancak authorization server, bu token'ı doğrulayıp yeni bir access token sağlayabilir.
4. **Verilen Yetkiler:**
    - Refresh token, istemciye **resource owner (kaynak sahibi)** tarafından verilen yetkileri temsil eder ve bu token, access token ile aynı yetkileri taşıyan yeni token'ların alınmasına imkan tanır.
5. **Yeniden Kullanım:**
    - **Refresh token** yalnızca authorization server ile etkileşimde kullanılabilir. Bu token, **resource server** ile paylaşılmaz. Refresh token, genellikle uzun süre geçerli olabilir, ancak authorization server, ihtiyaç durumuna göre yeni bir refresh token verebilir.



Refresh Token (Yenileme Token'ı)

Refresh token (yenileme token'ı), **access token**'ın süresi dolduğunda veya geçersiz hale geldiğinde yeni bir **access token** almak için kullanılan bir kimlik doğrulama aracıdır. Refresh token, istemciye **authorization server (yetkilendirme sunucusu)** tarafından verilir ve **resource server (kaynak sunucu)** ile doğrudan kullanılmaz. Refresh token, istemcinin sürekli olarak erişim sağlamak için yeni **access token**'lar alabilmesine imkan tanır.

Refresh Token'ın Temel Özellikleri:

1. **Yeni Access Token Alma:**
    - Refresh token, geçerli bir **access token**'ın süresi dolduğunda veya token geçersiz hale geldiğinde yeni bir access token almak için kullanılır. Access token'ın süresi genellikle kısa olur, bu yüzden refresh token ile yeni token almak gerekir.
2. **Sadece Yetkilendirme Sunucusunda Kullanım:**
    - Refresh token yalnızca **authorization server** ile etkileşime girmekte kullanılabilir. **Resource server** (kaynak sunucu) refresh token'ı almaz ve bu token herhangi bir şekilde bu sunuculara iletilmez.
3. **Genellikle Opak Yapı:**
    - Refresh token genellikle istemci için **opak** (şeffaf olmayan) bir dizedir. İstemci, token'ın içeriğini doğrudan göremez; ancak authorization server, bu token'ı doğrulayıp yeni bir access token sağlayabilir.
4. **Verilen Yetkiler:**

- Refresh token, istemciye **resource owner (kaynak sahibi)** tarafından verilen yetkileri temsil eder ve bu token, access token ile aynı yetkileri taşıyan yeni token'ların alınmasına imkan tanır.

5. **Yeniden Kullanım:**
    - **Refresh token** yalnızca authorization server ile etkileşimde kullanılabilir. Bu token, **resource server** ile paylaşılmaz. Refresh token, genellikle uzun süre geçerli olabilir, ancak authorization server, ihtiyaç durumuna göre yeni bir refresh token verebilir.

Refresh Token Akışı (Şekil 2):

Refresh token'ın kullanımını anlamak için aşağıdaki akışa göz atalım:

1. **Adım A:**
    - **Client (İstemci)**, authorization server'a bir **authorization grant** ile başvurarak bir access token talep eder.
2. **Adım B:**
    - Authorization server, istemciyi kimlik doğrulaması yaparak ve authorization grant'ı doğrulayarak bir **access token** ve **refresh token** verir.
3. **Adım C:**
    - İstemci, access token'ı kullanarak **resource server**'a korunan kaynağa erişim talep eder.
4. **Adım D:**
    - **Resource server** access token'ı doğrular ve geçerliyse kaynakla ilgili işlemi gerçekleştirir.
5. **Adım E:**
    - Adım C ve D tekrarlanır. Access token'ın süresi dolana kadar bu adımlar devam eder.
6. **Adım F:**
    - Access token süresi dolmuşsa, **resource server** geçersiz token hatası döndürür.
7. **Adım G:**
    - İstemci, **refresh token**'ı kullanarak authorization server'a başvurarak yeni bir access token talep eder.
8. **Adım H:**
    - Authorization server, refresh token'ı doğrular ve geçerliyse yeni bir access token (ve isteğe bağlı olarak yeni bir refresh token) verir.

Refresh Token ile Erişim Token'ı Yenileme Süreci:

- **Step (A) to (B):** İstemci bir authorization grant ile **access token** ve **refresh token** almak için authorization server'a başvurur.
- **Step (C) to (D):** İstemci, **access token**'ı kullanarak protected resource'a erişim talep eder. **Resource server**, token'ı doğrular ve geçerli ise erişim izni verir.

- **Step (E) to (F): Access token** süresi dolarsa, resource server geçersiz token hatası verir.
- **Step (G) to (H):** İstemci, **refresh token** ile yeni bir **access token** alır.

Refresh Token’ın Avantajları:

1. **Sürekli Erişim:**
    - Refresh token, istemcinin kesintisiz bir şekilde kaynaklara erişmesini sağlar. Bir kere kullanıldığında istemci, sürekli olarak yeni access token'lar alarak kaynağa erişebilir.
2. **Token Yönetimini Kolaylaştırma:**
    - Access token'ların geçerlilik sürelerinin kısa tutulması güvenliği artırır, ancak refresh token kullanılarak istemci sürekli olarak yeni token alabilir. Bu, güvenliği sağlamanın yanı sıra token yönetimini daha verimli hale getirir.
3. **İleri Düzey Güvenlik:**
    - **Access token**'ların kısa süreli geçerliliği, potansiyel kötüye kullanım durumlarında riskleri azaltır. **Refresh token** yalnızca authorization server ile kullanıldığından, istemci ve resource server arasındaki güvenlik seviyesi artırılır.

Refresh Token Kullanımı ve Güvenlik:

- Refresh token'lar güvenli bir şekilde saklanmalıdır, çünkü **resource server** ile hiç paylaşılmazlar ve bu token'lar ile yeni **access token** alınabilir.
- **Authorization server** güvenlik politikalarına göre refresh token'ın süresini ve geçerliliğini yönetebilir. Örneğin, bir **refresh token** yalnızca belirli bir süre sonra geçerliliğini yitirebilir.

Sonuç:

Refresh token, OAuth 2.0 protokolünde istemcilerin **access token**'larının süresi dolduğunda veya geçersiz hale geldiğinde yeni bir token almak için kullandığı önemli bir araçtır. Bu token, **authorization server** ile etkileşimde bulunarak yeni **access token**'lar alınmasına olanak tanır ve sürekli bir erişim sağlar. Refresh token'ın yalnızca **authorization server** ile kullanılması, güvenlik açısından önemli bir avantaj sunar.

TLS Version (Transport Layer Security Sürümü)

TLS (Transport Layer Security), internet üzerinde güvenli veri iletimi sağlamak için kullanılan bir protokoldür. Bu protokol, verilerin şifrelenmesi ve doğruluğunun sağlanması amacıyla kullanılır.

Açıklama:

1. **TLS Sürümünün Zamanla Değişmesi:**
    - TLS sürümleri zaman içinde gelişir ve daha güvenli hale gelir. Bu nedenle, kullanılan TLS sürümü, zamanla değişebilir ve **bilinen güvenlik açıklarına göre güncellenir**.
    - Bu belirli OAuth 2.0 spesifikasyonunda, **TLS kullanımı** belirli bir sürümle sınırlı olmayıp, **uygulamaların ve ortamların ihtiyaçlarına göre değişebilir**.
2. **TLS 1.2'nin Durumu:**
    - Yazının yazıldığı sırada, **TLS 1.2** en son sürüm olarak belirtilmiştir. **TLS 1.2**, **RFC5246** belgesine dayanmaktadır. Ancak, bu sürümün yaygın kullanımı sınırlıdır ve tüm ortamlar veya uygulamalar için uygun olmayabilir. Bu, özellikle bazı eski sistemlerde TLS 1.2'nin desteklenmediği anlamına gelebilir.
3. **TLS 1.0 ve Yaygın Kullanımı:**
    - **TLS 1.0** (RFC2246) daha yaygın olarak kullanılan bir sürümdür. Bu sürüm, birçok sistem ve uygulama tarafından geniş bir şekilde desteklenir, bu nedenle geniş **interoperabilite (uyumluluk)** sağlar. Ancak, güvenlik açıkları nedeniyle artık eski ve daha güvenli alternatifler kullanılmaktadır.
4. **Ekstra Güvenlik Mekanizmaları:**
    - Spesifikasyona göre, uygulamalar sadece TLS değil, **ekstra güvenlik önlemleri** veya protokoller de kullanabilirler. Bu, daha özel güvenlik gereksinimlerine sahip uygulamalar için ek protokoller veya şifreleme yöntemleri sunar.

Özet:

Bu bölüm, **TLS protokolü** ile ilgili iki ana noktayı vurgulamaktadır:

- **TLS 1.2**, şu anki yazımda en son sürüm olarak belirtilmiş olmasına rağmen, yaygın olarak desteklenmemektedir ve bazı uygulamalar için kullanımda zorluklar olabilir.
- **TLS 1.0** daha yaygın bir şekilde kullanılmaktadır, ancak eski ve daha güvenli alternatiflere göre güvenlik zaafiyetleri taşıyabilir.
- Ayrıca, **ekstra güvenlik önlemleri** uygulanabilir ve kullanılan sürüm zamanla değişebilir.

Sonuç olarak, güvenlik gereksinimleri ve çevresel koşullara bağlı olarak TLS sürümü seçilmelidir.

HTTP Redirections bölümü, OAuth 2.0 protokolünde HTTP yönlendirmelerinin nasıl kullanıldığını açıklamaktadır. Bu yönlendirmeler, istemcinin veya yetkilendirme sunucusunun, kaynak sahibinin (resource owner) kullanıcı aracını (user-agent) başka bir hedefe yönlendirmesi sürecidir.

HTTP Yönlendirmeleri (Redirections) Nedir?

HTTP yönlendirmeleri, bir kaynağa yapılan bir isteğin başka bir URL'ye yönlendirilmesi işlemidir. Yönlendirme, HTTP protokolü üzerinden belirli bir **HTTP durum kodu** (status code) ile yapılır. Bu protokolde yönlendirme işlemi **302** durum kodu ile genellikle yapılır, ancak başka durum kodları ve yönlendirme yöntemleri de kullanılabilir.

OAuth 2.0'da HTTP Yönlendirmelerinin Kullanımı

OAuth 2.0 protokolü, yönlendirmeleri **istemci (client)** ile **kaynak sahibinin kullanıcı aracı (user-agent)** arasında iletişimi sağlamak için yaygın şekilde kullanır. Aşağıda bu yönlendirmelerin genel bir akışı bulunmaktadır:

1. **Yetkilendirme İsteği**: İstemci, kullanıcıdan yetkilendirme almak için bir **yetkilendirme isteği** yapar. Bu istek, kullanıcının tarayıcısına (user-agent) yönlendirilir.
2. **Yetkilendirme Sunucusu**: Yetkilendirme sunucusu, kullanıcının kimliğini doğrular ve ardından yönlendirme gerçekleştirilir. Yönlendirme, kullanıcının tarayıcısını (user-agent) bir sonraki adıma, yani istemciye doğru gönderir.
3. **Yönlendirme ve İzin Verme**: Kullanıcı, yetkilendirme sunucusundan aldığı yönlendirmeyi takip eder ve istemciye gerekli yetkilendirmeyi verir.
4. **Yönlendirme Sonucu**: Bu yönlendirme, HTTP durum kodu 302 gibi bir yönlendirme kodu ile yapılır. Yönlendirme, belirli bir URL’ye gitmesi gerektiğini belirtir ve kullanıcı aracını ilgili yere yönlendirir.

302 Durum Kodu

- **302**: Bu HTTP durum kodu, isteğin geçici olarak başka bir yere yönlendirilmesi gerektiğini belirtir. OAuth 2.0'da bu, **yetkilendirme kodu** veya **erişim belirteci (access token)** gibi bir yanıtın istemciye verilmesi için kullanılır.

Yönlendirme Protokolü

OAuth 2.0 protokolü, HTTP yönlendirmeleri aracılığıyla istemci ve yetkilendirme sunucusu arasında bir etkileşim sağlar. Ancak bu, yalnızca istemcilerin kullanıcının kimlik doğrulaması ve yetkilendirme için kullandığı bir yöntemdir.

Redirection'ların avantajları:

- **Kolay Entegrasyon**: Kullanıcı kimliği doğrulandıktan sonra istemciye verilecek belirteçlerin (token) güvenli bir şekilde yönlendirilmesi sağlanır.
- **Güvenlik**: Yönlendirme sırasında, kaynak sahibinin (kullanıcının) kimlik bilgileri istemciyle paylaşılmadan sadece yetkilendirme sunucusunda doğrulanır.

Özet

- **HTTP Yönlendirmeleri**, istemci ve kullanıcı arasındaki etkileşimde önemli bir rol oynar.

- OAuth 2.0 protokolünde, bu yönlendirmeler, kullanıcı aracını (tarayıcıyı) doğru adrese yönlendirmek için kullanılır.
- 302 durumu, en yaygın kullanılan yönlendirme türüdür, ancak OAuth 2.0, farklı HTTP yönlendirme kodlarını ve tekniklerini de kullanabilir.

Bu yönlendirmeler, OAuth protokolünün güvenli ve etkili çalışabilmesi için kritik bir bileşendir.

1.8. Interoperability bölümü, OAuth 2.0 protokolünün sağlamış olduğu güçlü yetkilendirme çerçevesine rağmen, protokolün farklı uygulamalar arasında tamamen uyumlu bir şekilde çalışmasının bazen zor olabileceğini belirtmektedir. Burada anlatılan ana noktalar şunlardır:

1. OAuth 2.0'un Zengin ve Esnek Yapısı

OAuth 2.0, esnek ve genişletilebilir bir yetkilendirme çerçevesi sunar. Bu, çeşitli kullanım senaryoları ve ihtiyaçlar için uyarlanabilir hale gelmesini sağlar. Ancak bu esneklik, aynı zamanda **uyumsuzluk** sorunlarına yol açabilir. Yani, bir OAuth 2.0 uygulamasının diğer OAuth 2.0 uygulamalarıyla doğru şekilde çalışabilmesi, standarttan çok fazla sapmamalıdır. Aksi halde, farklı OAuth 2.0 implementasyonları birbirleriyle uyumsuz olabilir.

2. Eksik ve Tanımlanmamış Bileşenler

OAuth 2.0, bazı gerekli bileşenleri veya özellikleri tam olarak tanımlamamaktadır. Bu eksiklikler, özellikle aşağıdaki bileşenleri içerir:

- **Client Registration (İstemci Kaydı)**: İstemcilerin yetkilendirme sunucusuna nasıl kaydedileceği hakkında net bir yönerge bulunmamaktadır.
- **Authorization Server Capabilities (Yetkilendirme Sunucusu Yetenekleri)**: Yetkilendirme sunucularının hangi özellikleri desteklemesi gerektiği tam olarak tanımlanmamıştır.
- **Endpoint Discovery (Uç Nokta Keşfi)**: Yetkilendirme ve kaynak sunucularının uç noktalarını keşfetme yöntemleri konusunda eksiklikler vardır.

Bu eksiklikler, OAuth 2.0 implementasyonlarının birbirleriyle uyumlu çalışmasını zorlaştırabilir. İstemciler, belirli bir yetkilendirme sunucusuyla ve kaynak sunucusuyla doğru şekilde iletişim kurabilmek için elle ve özel olarak yapılandırılmalıdır.

3. Gelecekteki Çalışmalar ve Uyum Sağlama

OAuth 2.0'ın mevcut hali, uyumluluğu artırmak için daha fazla **profil tanımlamayı** ve **uzantılar geliştirmeyi** gerektirecektir. Bu eksikliklerin ve potansiyel uyumsuzlukların üstesinden gelmek için gelecekte yapılacak çalışmaların, web ölçeğinde **tam uyumluluğu** sağlayacak daha fazla standart ve yönerge tanımlaması beklenmektedir.

Özetle

OAuth 2.0, esnekliği nedeniyle farklı uygulamalar için uyarlanabilir, ancak bu esneklik bazı durumlarda farklı uygulamaların birbirleriyle uyumsuz olmasına yol açabilir. Ayrıca, protokolün bazı bileşenleri tam olarak tanımlanmadığı için, istemcilerin ve sunucuların birbirleriyle uyumlu olabilmesi için manuel yapılandırmalar gerekebilir. Gelecekte, bu uyumsuzlukları gidermek amacıyla daha fazla standart ve profil tanımlanması beklenmektedir.

1.9. Notational Conventions bölümü, OAuth 2.0 belgesinde kullanılan bazı dilbilgisel kuralları ve özel terimleri açıklamaktadır. Bu bölümde geçen kavramlar, belgenin daha net ve tutarlı anlaşılmasını sağlamak için belirli kurallara dayanmaktadır. İşte önemli noktalar:

1. Anahtar Kelimeler (Key Words)

Bu bölümde geçen anahtar kelimeler, RFC 2119'a (Standartlara Yönelik Belgelerde Kullanılacak Terimler) dayanarak açıklanır. Bu terimler, protokoldeki zorunlulukların, önerilerin ve seçeneklerin anlamını netleştirmek için kullanılır:

- **MUST / REQUIRED / SHALL**: Bir şeyin **zorunlu** olduğunu belirtir. Bu, belirli bir davranışın uygulanması gerektiği anlamına gelir.
- **MUST NOT / SHALL NOT**: Bir şeyin **yapılmaması gerektiğini** belirtir. Yani, bu davranış yasaktır.
- **SHOULD / RECOMMENDED**: Bir şeyin **önerildiğini** belirtir. Yani, ideal olarak yapılması gereken bir şeydir, ancak zorunlu değildir.
- **SHOULD NOT**: Bir şeyin **yapılmaması önerildiğini** belirtir. Ancak, bunu yapmak yasak değildir.
- **MAY / OPTIONAL**: Bir şeyin **isteğe bağlı** olduğunu belirtir. Yani, bunu yapmak serbesttir.

2. ABNF Notasyonu

OAuth 2.0 belgesinde kullanılan dilbilgisel yapılar, **ABNF (Augmented Backus-Naur Form)** notasyonuyla ifade edilir. Bu, protokolün kurallarını ve yapılarını tanımlamak için kullanılan bir dilbilgisel formattır ve RFC 5234'te açıklanmıştır. ABNF, metin ve parametrelerin nasıl yazılacağına dair kurallar sunar.

3. URI Referansları

OAuth 2.0, **URI referanslarını** kullanır (Uniform Resource Identifier). URI'ler, web üzerindeki kaynakları tanımlamak için kullanılan standartlardır ve bu belge, URI'lerin nasıl kullanılacağını belirten kuralları içerir. Bu konuyla ilgili daha fazla detay RFC 3986'da bulunabilir.

4. Güvenlik İlgili Terimler

OAuth 2.0'da, güvenlikle ilgili bazı terimler, **RFC 4949**'da tanımlandığı şekilde anlaşılmalıdır. Bu terimler şunlardır (bunlarla sınırlı olmamak kaydıyla):

- **Attack**: Saldırı
- **Authentication**: Kimlik doğrulama
- **Authorization**: Yetkilendirme
- **Certificate**: Sertifika
- **Confidentiality**: Gizlilik
- **Credential**: Kimlik bilgisi
- **Encryption**: Şifreleme
- **Identity**: Kimlik
- **Sign / Signature**: İmzalama / İmza
- **Trust**: Güven
- **Validate / Verify**: Doğrulama / Onaylama

Bu terimler, güvenlik bağlamında net bir anlayış sağlamak için kullanılır.

5. Büyük/Küçük Harf Duyarlılığı

OAuth 2.0 belgesindeki tüm protokol parametre adları ve değerleri **büyük/küçük harf duyarlıdır**. Yani, örneğin "Token" ve "token" farklı parametreler olarak kabul edilir.

Özetle

Bu bölüm, OAuth 2.0 spesifikasyonunda kullanılan dil ve terimler için belirli kurallar koyar. **Anahtar kelimeler** belirli zorunluluklar veya öneriler ifade eder, **ABNF notasyonu** dilbilgisel kuralları tanımlar, **güvenlik terimleri** belirli bir anlamda anlaşılmalıdır ve **büyük/küçük harf duyarlılığı** parametre adları ve değerleri için geçerlidir. Bu kurallar, belgenin doğru şekilde anlaşılmasını ve uygulamaların doğru şekilde yorumlanmasını sağlar.