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.

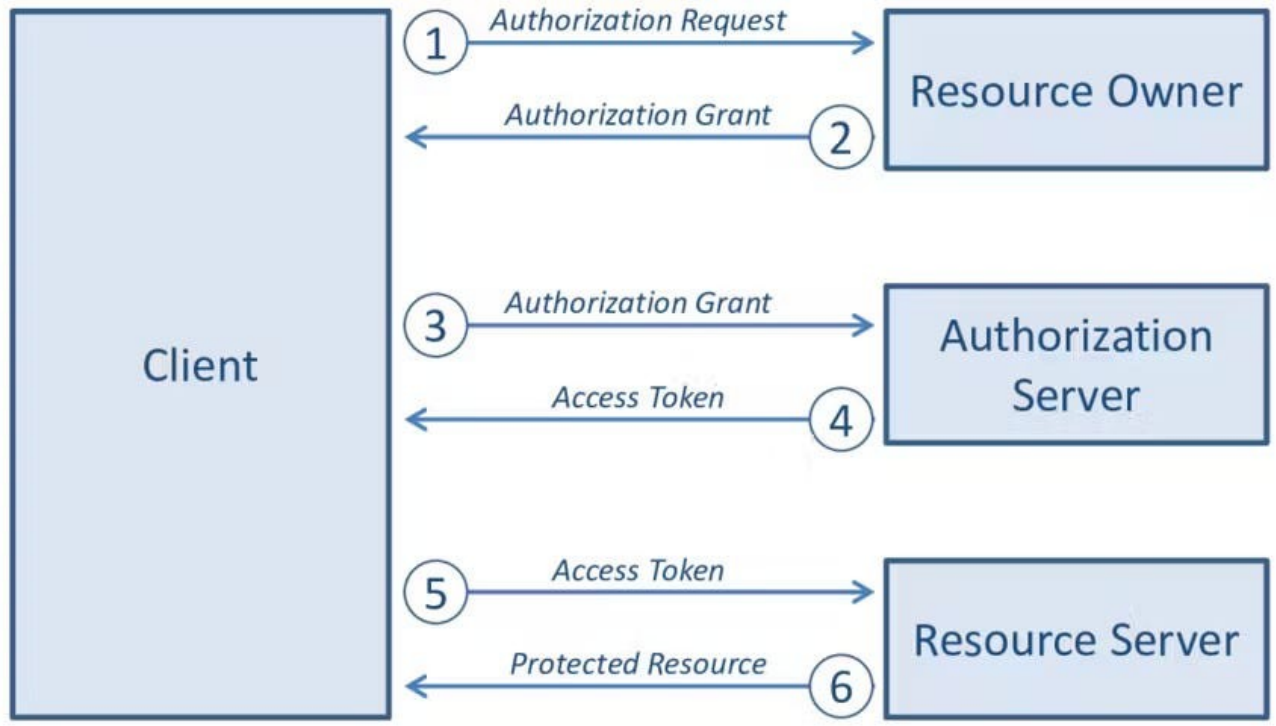
Özet

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:

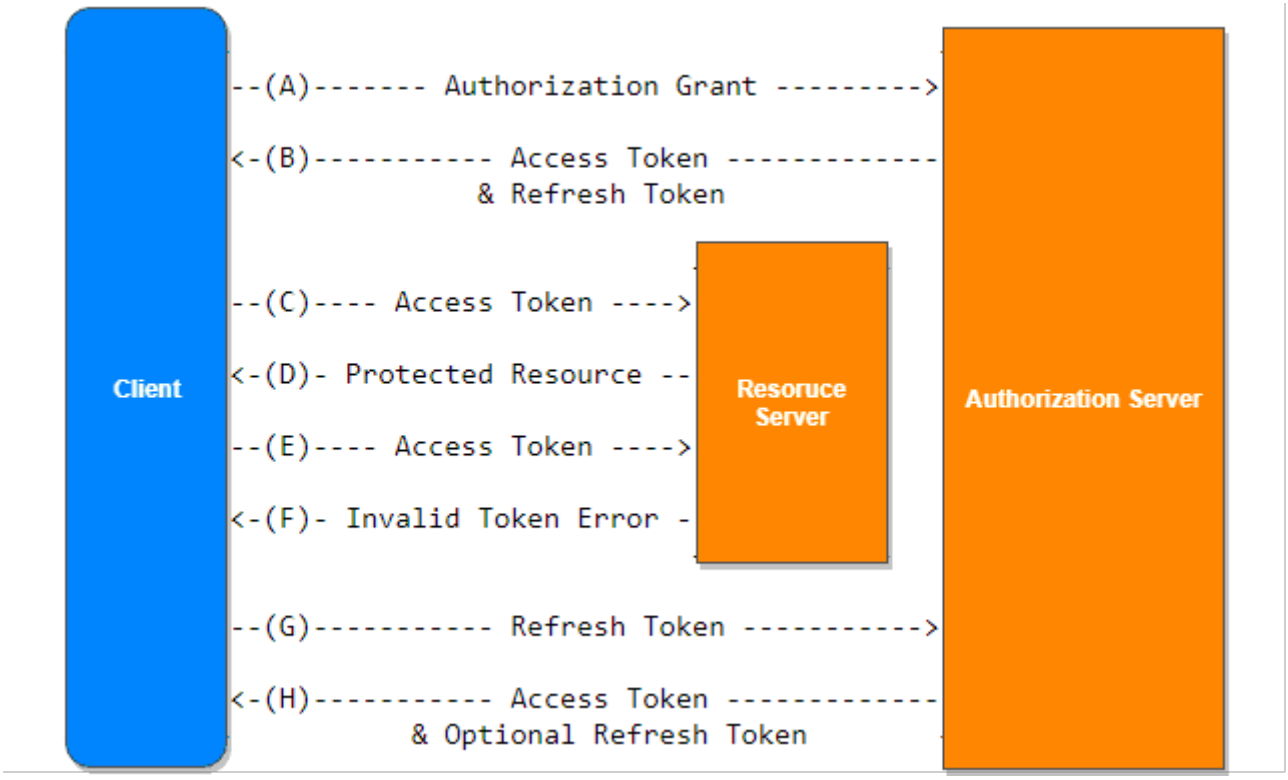
Access token, OAuth 2.0 protokolünde istemcilerin, kaynaklara erişim izni almasını sağlayan bir kimlik doğrulama aracıdır. Token, erişim hakkı tanımlayan ve sınırlayan bir dizedir ve istemci tarafından kullanılarak korunan kaynaklara erişim sağlanır. Token'ın yapısı ve güvenlik seviyesi, **resource server**'ın 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.

2. Client Registration bölümü, OAuth 2.0 protokolünün başlatılmasından önce bir **client** (istemci) uygulamasının yetkilendirme sunucusuna nasıl kaydolması gerektiğini açıklar. Bu kayıt süreci, istemcinin yetkilendirme sunucusuyla etkileşimde bulunmasını gerektirir. Ancak, bu etkileşimlerin ne şekilde gerçekleşeceği ve kaydın nasıl yapılacağı, OAuth 2.0 spesifikasyonunun kapsamı dışında kalır. Ancak genellikle, bu süreç bir HTML kayıt formu aracılığıyla bir kullanıcı etkileşimi gerektirir.

1. İstemci Kayıt Süreci

İstemci kaydının gerekliliği, OAuth 2.0 protokolünü başlatmadan önce istemcinin yetkilendirme sunucusuna kaydolmasını zorunlu kılar. Ancak, kayıt sürecinde istemcinin doğrudan yetkilendirme sunucusuyla etkileşimde bulunması her zaman gerekmez. Yetkilendirme sunucusu, kaydı başka yollarla da yapabilir. Örneğin, istemcinin güvenilir bir kanal aracılığıyla bir **self-issued** (kendiliğinden verilen) veya **third-party-issued** (üçüncü tarafça verilen) bir beyanname sunması, ya da istemcinin kaydını gerçekleştirmek için bir **client discovery** (istemci keşfi) süreci uygulanabilir.

Bu, istemci kayıt işleminin, bazı durumlarda doğrudan etkileşim gerektirmeden de gerçekleştirilebileceği anlamına gelir.

2. Kayıt Sürecinde İstemcinin Sağlaması Gereken Bilgiler

İstemci kaydını gerçekleştiren geliştirici, aşağıdaki bilgileri sağlamalıdır:

- **İstemci tipi**: İstemci kaydında, istemcinin tipi belirtilmelidir. Bu, OAuth 2.0 protokolü için çok önemlidir çünkü istemcinin güvenlik özellikleri, protokolde nasıl işlem yapacağı ve hangi tür verilere erişim talep edeceği, istemci tipine göre değişir.
- **Redirection URIs**: Kayıt sırasında, istemci uygulaması tarafından kullanılan **redirection URIs** (yönlendirme URI'ları) belirtilmelidir. Bu URI'lar, istemci uygulamasının yetkilendirme sunucusundan alacağı yanıtların yönlendirilmesi gereken yerlerdir. Bu bilgiler, istemci ile sunucu arasındaki iletişimde güvenliği sağlamak için kritik öneme sahiptir.
- **Diğer gerekli bilgiler**: İstemci kaydı için, yetkilendirme sunucusunun talep edebileceği diğer bilgiler de sağlanmalıdır. Örneğin, istemci uygulamasının adı, web sitesi, açıklaması, logosu ve yasal şartları kabul etme gibi bilgiler de istenebilir.

3. Güven ve Güvenlik

Kaydın güvenli bir şekilde yapılması için, istemci ve yetkilendirme sunucusu arasında güvenli bir kanal üzerinden etkileşim sağlanmalıdır. Bu, istemci bilgilerinin doğru bir şekilde alınıp kaydedilmesini ve doğru istemci tipinin ve diğer bilgilerin doğrulanmasını sağlar.

Özetle

Client Registration, istemci uygulamasının OAuth 2.0 protokolüne katılabilmesi için yetkilendirme sunucusuna kaydolmasını gerektirir. Bu kayıtta, istemcinin tipi, redirection URI'ları gibi bilgilerin yanı sıra diğer güvenlik önlemleri de dikkate alınır. Ancak, doğrudan istemci-yetkilendirme sunucusu etkileşimi her zaman gerekli değildir ve farklı güvenli yöntemlerle istemci kaydı yapılabilir.

2.1. Client Types bölümü, OAuth 2.0 protokolünde istemcilerin iki ana tipte sınıflandırıldığını açıklar. Bu sınıflandırma, istemcilerin yetkilendirme sunucusuyla güvenli bir şekilde kimlik doğrulaması yapabilme yeteneklerine dayanır. Temelde, istemcilerin kimlik bilgilerini (client credentials) gizli tutma kapasitesine göre iki ana türü belirlenmiştir:

1. Confidential Clients (Gizli İstemciler)

Confidential istemciler, kimlik bilgilerini güvenli bir şekilde saklayabilen ve yetkilendirme sunucusu ile güvenli bir şekilde kimlik doğrulaması yapabilen istemcilerdir. Bu istemciler, genellikle güvenli bir sunucu üzerinde çalışır ve istemci kimlik bilgilerine yalnızca kısıtlı erişimi olan uygulamalar tarafından erişilebilir.

- Örnek: Web uygulamaları, API istemcileri veya sunucu tarafı uygulamaları.
- Güvenlik: Bu istemciler, kimlik bilgilerini (client credentials) saklamak ve bunları güvenli bir şekilde kullanmak için gereken altyapıya sahiptir. Bu nedenle, bu tür istemciler daha güvenli kabul edilir.

2. Public Clients (Açık İstemciler)

Public istemciler, kimlik bilgilerini güvenli bir şekilde saklayamayan istemcilerdir. Bu istemciler, genellikle kullanıcının cihazında çalışan uygulamalardır (örneğin, yerel uygulamalar veya web tarayıcısında çalışan uygulamalar). Bu tür istemciler, kimlik bilgilerini güvenli bir şekilde saklayamazlar çünkü istemci kodu, kullanıcıya açık bir şekilde dağıtılır ve kullanıcıya veya cihazdaki diğer uygulamalara erişilebilir.

- Örnek: Yerel uygulamalar (native applications), web tarayıcı tabanlı uygulamalar.
- Güvenlik: Bu istemcilerde, kimlik bilgileri genellikle uygulama içinde açıkça saklanır, bu da güvenlik risklerine yol açabilir. Bu tür istemciler, genellikle kimlik doğrulaması için başka güvenlik önlemleri (örneğin, kullanıcının kimliği doğrulandıktan sonra kısa süreli erişim jetonları kullanma) gerektirir.

3. Client Type Belirlenmesi

İstemci türü, yetkilendirme sunucusunun güvenli kimlik doğrulama tanımına ve istemci kimlik bilgilerini açıklamaya karşı kabul edilebilir seviyesine dayanır. Yetkilendirme sunucusu, istemci türü konusunda varsayımlar yapmamalıdır, yani her istemci türünün güvenlik gereksinimleri farklıdır ve her birine uygun şekilde yaklaşılmalıdır.

4. Dağıtık İstemciler

Bir istemci, birden fazla bileşen içeriyor olabilir ve her bileşenin farklı bir istemci türü ve güvenlik bağlamı olabilir. Örneğin, bir dağıtık istemci hem gizli istemci bileşenlerine (sunucu tabanlı) hem de açık istemci bileşenlerine (tarayıcı tabanlı) sahip olabilir. Eğer yetkilendirme sunucusu bu tür istemcileri desteklemiyorsa, her bileşen ayrı bir istemci olarak kaydedilmelidir.

5. OAuth 2.0 İstemci Profilleri

OAuth 2.0, farklı istemci türleri için spesifik profiller tanımlar:

- **Web Application (Web Uygulaması)**: Bu, bir **confidential client** (gizli istemci) olup, web sunucusunda çalışır. Kaynak sahipleri, bu istemciye HTML kullanıcı arayüzü aracılığıyla

erişir. İstemci kimlik bilgileri ve erişim jetonları yalnızca web sunucusunda saklanır ve kaynak sahibine veya diğer kullanıcı ajanlarına (örneğin, tarayıcılar) erişilebilir değildir.

- **User-Agent-Based Application (Kullanıcı Ajansı Tabanlı Uygulama)**: Bu, bir **public client** (açık istemci) olup, istemci kodu bir web sunucusundan indirilir ve kullanıcı ajanı (örneğin, web tarayıcısı) üzerinde çalıştırılır. Bu tür istemcilerde, kimlik bilgileri ve protokol verileri kaynak sahibi tarafından kolayca erişilebilir ve genellikle görünürdür. Ancak, kullanıcı ajanının özellikleri, yetkilendirme talebinde bulunurken yardımcı olur.
- **Native Application (Yerel Uygulama)**: Bu da bir **public client** olup, kaynak sahibinin cihazına kurulur ve çalıştırılır. Bu tür istemcilerde, protokol verileri ve kimlik bilgileri erişilebilir, ancak uygulama tarafından kullanılan dinamik olarak verilen kimlik bilgileri (örneğin, erişim jetonları) kabul edilebilir seviyede korunabilir. Bu tür kimlik bilgileri, kötü niyetli sunuculardan korunduğu gibi, bazen cihazdaki diğer uygulamalardan da korunabilir.

6. Özet

OAuth 2.0, istemcileri **gizli** ve **açık** olarak iki ana kategoride sınıflandırır. **Gizli istemciler**, kimlik bilgilerini güvenli bir şekilde saklayabilirken, **açık istemciler** bunu yapamaz ve genellikle daha yüksek güvenlik önlemleri gerektirir. OAuth 2.0, farklı istemci türleri için güvenlik gereksinimlerini ve uygulama senaryolarını dikkate alarak, her tür için uygun çözümler sunar.

2.2. Client Identifier bölümü, OAuth 2.0 protokolünde istemcinin yetkilendirme sunucusundan aldığı "client identifier" (istemci kimliği) hakkında bilgi verir. Bu kimlik, istemcinin kaydının benzersiz bir temsili olarak kullanılır ve bir dizi önemli özellik taşır.

1. Client Identifier Nedir?

Client identifier, istemcinin kaydını temsil eden benzersiz bir dizidir. Yetkilendirme sunucusu, istemci kaydını doğruladıktan sonra, her istemciye bu tür bir kimlik verir. Bu kimlik, istemcinin **yetkilendirme sunucusu** ile olan ilişkisinin bir parçası olarak kullanılır ve istemcinin kaydını tanımlar.

2. Client Identifier'ın Gizli Olmaması

Client identifier, **gizli bir bilgi değildir**. Bu kimlik, **kaynak sahibine** (resource owner) açık bir şekilde gösterilebilir ve istemci ile kaynak sahibi arasında karşılaşılan herhangi bir yerde görünebilir. Bu nedenle, client identifier yalnızca istemciyi tanımlamak için kullanılır ve **client authentication (istemci kimlik doğrulama)** amacıyla tek başına kullanılamaz.

Örneğin, istemci kimliği genellikle istemci tarafından yapılan isteklerde başlıklar (headers) veya URL parametreleri gibi açık yollarla gönderilebilir, çünkü bu kimlik **gizli** değildir ve her zaman gizlilik veya güvenlik için saklanmaz.

3. Benzersizlik ve Yetkilendirme Sunucusuna Bağlılık

Client identifier, her yetkilendirme sunucusu için **benzersiz** olmalıdır. Yani, bir istemci için atanan kimlik, sadece o yetkilendirme sunucusu tarafından geçerli olacaktır. Aynı istemci, başka bir yetkilendirme sunucusuyla kaydolduğunda, o sunucu ona farklı bir client identifier atayabilir. Bu, farklı sunucuların birbirlerinin istemci kimliklerini tanımayacaklarını ve bağımsız olarak çalışacaklarını garanti eder.

4. Kimlik Boyutu

Client identifier'ın boyutu, bu spesifikasyonda açıkça tanımlanmamıştır. Bu nedenle, istemciler bu kimliğin boyutu hakkında herhangi bir varsayımda bulunmamalıdır. Ancak, **yetkilendirme sunucusu**, istemciye verdiği kimliğin boyutunu **belgelendirmelidir**. Bu, istemcilerin doğru bir şekilde işlem yapabilmesi için önemlidir.

5. Kimlik Doğrulama İçin Kullanılamaz

Client identifier, istemcinin kimliğini belirlemek için kullanılsa da, **istemci doğrulama** için yeterli bir bilgi değildir. Yani, istemci kimliği yalnızca istemcinin kaydını tanımlar ve istemcinin yetkilendirme sunucusuyla güvenli bir şekilde iletişim kurabilmesi için ek güvenlik önlemleri gereklidir (örneğin, istemci şifresi veya başka bir kimlik doğrulama yöntemi).

6. Özet

- **Client identifier**, istemcinin kaydını temsil eden benzersiz bir dizidir ve yalnızca istemcinin tanımlanmasında kullanılır.
- Bu kimlik **gizli değildir**, kaynak sahibi tarafından görülebilir ve istemci kimlik doğrulamasında tek başına kullanılmaz.

- Kimlik, her yetkilendirme sunucusu için benzersizdir ve istemci bu kimlik ile ilişkilendirilen sunucuya bağlıdır.
- Kimlik boyutu belirsizdir ancak yetkilendirme sunucusu bunu **belgelendirmelidir**.

Sonuç olarak, client identifier, istemciyi tanımlamak için kullanılan bir kimliktir ancak tek başına güvenlik için yeterli değildir ve doğrulama amacıyla kullanılmaz.

2.3. Client Authentication bölümü, istemci türü "confidential" (gizli) olan bir istemcinin, yetkilendirme sunucusuyla nasıl kimlik doğrulaması yapması gerektiğini açıklar. Bu bölümde, istemci doğrulaması için kullanılan yöntemlerin güvenlik gereksinimlerine dayalı olarak nasıl yapılandırılacağına dair önemli bilgiler bulunmaktadır.

1. Confidential Client (Gizli İstemci)

Confidential client, istemci kimlik bilgilerini güvenli bir şekilde saklayabilen bir istemci türüdür. Genellikle, istemci uygulaması, bir **sunucu** üzerinde çalışır ve istemci kimlik bilgileri (örneğin, parola, özel anahtar ve kamu anahtar çifti gibi) bu sunucuda güvenli bir şekilde saklanır.

- **Gizli İstemciler için Kimlik Doğrulama Yöntemleri**: Yetkilendirme sunucusu, gizli istemci türü için güvenlik gereksinimlerine uygun bir **kimlik doğrulama yöntemi** belirler. Bu, şifre tabanlı bir kimlik doğrulama, dijital sertifikalar veya kamu/özel anahtar çifti gibi farklı yöntemler olabilir. Yetkilendirme sunucusu, istemcinin güvenli bir şekilde kimliğini doğrulamak için bu kimlik doğrulama yöntemlerini kabul edebilir. Bu, istemcinin kimlik doğrulaması yapılırken sunucu ve istemci arasındaki güvenli iletişimi sağlar.
- **Client Credentials**: Gizli istemciler, genellikle bir **set** (takım) istemci kimlik bilgisi (client credentials) alır. Bu kimlik bilgileri, istemcinin yetkilendirme sunucusuyla güvenli bir şekilde kimliğini doğrulaması için kullanılır. Örneğin, istemci bir **parola** ya da **kamu/özel anahtar çifti** kullanabilir. Bu kimlik bilgileri istemciyle birlikte güvenli bir şekilde saklanır.

2. Public Client (Halka Açık İstemci)

Public client, istemci kimlik bilgilerini güvenli bir şekilde saklayamayan ve genellikle bir kullanıcının cihazında çalışan istemci türüdür. Örneğin, web tarayıcılarında veya mobil cihazlarda çalışan istemciler, istemci kimlik bilgilerini güvenli bir şekilde saklamakta zorluk yaşayabilirler. Bu nedenle, **public client** istemcileri için kimlik doğrulaması yapılmaz.

- Yetkilendirme sunucusu, **public client** istemcileri için kimlik doğrulama yöntemleri belirleyebilir. Ancak, public client kimlik doğrulaması **istemcinin tanımlanması** için kullanılmamalıdır. Yani, **public client** türündeki istemciler, kimlik doğrulama için **client authentication** yöntemlerine dayanmazlar.

3. Birden Fazla Kimlik Doğrulama Yöntemi Kullanılamaz

Bir istemci, aynı istek içinde birden fazla kimlik doğrulama yöntemi kullanmamalıdır. Bu, istemcinin güvenliğini ve iletişimini gereksiz yere karmaşıklaştırabilir ve sunucu için güvenlik açıklarına yol açabilir. Örneğin, istemci aynı istekte hem şifre hem de anahtar doğrulaması kullanamaz. İstemci yalnızca bir kimlik doğrulama yöntemi seçmeli ve bu yöntemi kullanmalıdır.

4. Özet

- **Confidential client** için, yetkilendirme sunucusu güvenlik gereksinimlerine uygun bir kimlik doğrulama yöntemi belirler ve istemci, bu yöntemle kimlik doğrulaması yapar.
- Gizli istemciler, genellikle bir set **client credentials** (kimlik bilgileri) alır ve bu bilgileri güvenli bir şekilde kullanır.
- **Public client** istemcileri, kimlik doğrulama için yalnızca token (erişim belirteci) gibi mekanizmalar kullanır, kimlik doğrulama amacıyla istemci bilgileri kullanılamaz.

- İstemci, her istekte yalnızca **bir** kimlik doğrulama yöntemi kullanabilir.

Bu bölüm, istemcinin, güvenli bir kimlik doğrulama süreciyle yetkilendirme sunucusuyla iletişim kurmasını ve doğru güvenlik gereksinimlerine dayalı olarak işlem yapmasını sağlar.

2.3.1. Client Password bölümü, istemcilerin kimlik doğrulamasını yapmak için **client password** (istemci parolası) kullanma yöntemini açıklar. Bu yöntem, istemcinin kimliğini doğrulamak için **HTTP Basic Authentication** şemasını kullanmayı içerir. Ayrıca, istemci kimlik bilgileriyle birlikte gönderilebilecek alternatif bir yöntem ve güvenlik önlemleri de tartışılmaktadır.

1. HTTP Basic Authentication

İstemciler, bir **client password** (istemci parolası) sahibi olduklarında, yetkilendirme sunucusuyla kimlik doğrulaması yapmak için **HTTP Basic Authentication** şemasını kullanabilirler. Bu şema, RFC 2617'de tanımlanmıştır.

- **Client Password Kullanımı**: İstemci, kendisini tanıtmak ve kimliğini doğrulamak için HTTP isteği içerisinde, **client identifier** (istemci kimlik bilgisi) ve **client password** (istemci parolası) bilgisini içerir. Bu bilgileri, "application/x-www-form-urlencoded" kodlama algoritması ile encode eder.
    - **Client Identifier**: İstemcinin benzersiz kimliğidir.
    - **Client Password**: İstemcinin gizli parolasıdır.

  Bu bilgilerin her biri, HTTP isteğinde uygun şekilde encode edilerek, **Authorization header** (Yetkilendirme başlığı) ile gönderilir. Örnek olarak:

Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

Burada, `Basic` *anahtar kelimesi, temel kimlik doğrulamasının kullanılacağını belirtir ve ardından client identifier ile client password bilgileri encode edilerek yer alır.*

- **Yetkilendirme Sunucusunun Desteklemesi**: Yetkilendirme sunucusu, client password kullanan istemciler için **HTTP Basic Authentication** şemasını desteklemek zorundadır. Bu, istemcinin kimlik doğrulaması yapabilmesi için gereklidir.

2. Alternatif Kimlik Doğrulama Yöntemi

Yetkilendirme sunucusu, istemci kimlik bilgilerini **request body** (istek gövdesi) içinde de alabilir. Bu, client password yerine, istemci kimlik bilgilerini aşağıdaki parametrelerle göndermeyi içerir:

- **client_id**: İstemciye, kayıt sırasında verilen benzersiz kimlik bilgisi.
- **client_secret**: İstemcinin gizli parolası.

Bu yöntemin kullanımı, özellikle HTTP Basic Authentication şemasını veya başka bir parola tabanlı kimlik doğrulama şemasını doğrudan kullanamayan istemciler için uygundur. Ancak, bu yöntemin **request URI** (istek URI’sı) içinde yer almaması gerektiği belirtiliyor, sadece isteğin gövdesinde kullanılmalıdır.

Örnek kullanım:

POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
&client_id=s6BhdRkqt3&client_secret=7Fjfp0ZBr1KtDRbnfVdmIw

3. TLS (Transport Layer Security) Zorunluluğu

Yetkilendirme sunucusu, istemci kimlik doğrulaması için parola tabanlı bir yöntem kullanıldığında, **TLS** (Transport Layer Security) protokolünün kullanılmasını zorunlu kılar. TLS, verilerin güvenli bir şekilde iletilmesini sağlar ve parola gibi hassas bilgilerin güvenliğini artırır. Bu, istemci kimlik bilgileri ile yapılan her istekte **TLS** kullanılması gerektiğini belirtir.

4. Brute Force (Kaba Kuvvet) Saldırılarına Karşı Koruma

Bu kimlik doğrulama yöntemi, **client password** (istemci parolası) içerdiği için, yetkilendirme sunucusu, kaba kuvvet saldırılarına karşı koruma sağlamak zorundadır. Kaba kuvvet saldırıları, kötü niyetli bir saldırganın parolaları tahmin etmeye çalışarak istemci kimlik doğrulamasını kırmaya çalıştığı saldırılardır. Yetkilendirme sunucusu, bu tür saldırılara karşı savunma mekanizmaları eklemelidir.

Özetle:

- **Client Password** yöntemi, istemcilerin **HTTP Basic Authentication** şemasını kullanarak kimlik doğrulaması yapmasına olanak sağlar. Bu şemada, istemci kimlik bilgileri (client identifier ve client password) HTTP isteği içerisinde gönderilir.
- Alternatif olarak, istemci kimlik bilgileri isteğin gövdesinde **client_id** ve **client_secret** parametreleri olarak da gönderilebilir, ancak bu yöntem genellikle önerilmez.
- **TLS** kullanımı zorunludur; bu, istemci parolasının güvenli bir şekilde iletilmesi için gereklidir.
- Ayrıca, sunucunun kaba kuvvet saldırılarına karşı savunmasız olmaması için gerekli güvenlik önlemleri alması gerekir.

Bu bölüm, istemcilerin güvenli bir şekilde kimlik doğrulaması yapmalarını sağlamak için önemli bilgiler ve güvenlik önlemleri sunar.

2.3.2. Other Authentication Methods *bölümü, yetkilendirme sunucularının istemci kimlik doğrulaması için* ***HTTP authentication*** *şemalarını esnek bir şekilde kullanabilmelerini açıklamaktadır. Bu bölümde, yetkilendirme sunucusunun* ***HTTP Basic Authentication*** *dışında başka kimlik doğrulama yöntemlerini de destekleyebileceği ve bu yöntemlerin nasıl uygulanması gerektiği belirtiliyor.*

1. Diğer Kimlik Doğrulama Yöntemleri (Other Authentication Methods)

Yetkilendirme sunucusu, **HTTP authentication** (HTTP kimlik doğrulama) için gereksinimlerini karşılayan herhangi bir uygun kimlik doğrulama şemasını destekleyebilir. Yani, sadece **HTTP Basic Authentication** değil, başka kimlik doğrulama yöntemleri de kullanılabilir.

MAY ifadesi, yetkilendirme sunucusunun ihtiyaca göre farklı kimlik doğrulama şemalarını kullanabileceği, ancak bunun zorunlu olmadığı anlamına gelir. Örneğin:

- **OAuth 2.0 Bearer Token** gibi daha modern ve güvenli kimlik doğrulama yöntemleri,
- **OAuth 2.0 Client Assertion** (istemci iddiaları) gibi alternatifler,
- **OAuth 2.0 JWT (JSON Web Token)** gibi teknolojiler,
- Diğer özel şemalar (örneğin, şirketin iç güvenlik standartlarına uygun özel şemalar) kullanılabilir.

2. İstemci Kimliği ve Kimlik Doğrulama Yöntemi Arasındaki Eşleştirme

Eğer yetkilendirme sunucusu, **diğer kimlik doğrulama yöntemlerini** kullanıyorsa, bu durumda **client identifier** (istemci kimliği) ile belirli bir kimlik doğrulama şeması arasında açık bir eşleştirme yapılması gerekmektedir. Yani, her istemci kaydı, kullanılan kimlik doğrulama şemasıyla ilişkilendirilmelidir.

- Örneğin, bir istemci kayıtlıysa ve bu istemci özel bir kimlik doğrulama yöntemi kullanacaksa, yetkilendirme sunucusu, bu istemci kaydını o yönteme uygun şekilde eşleştirmelidir.
- Bu eşleştirme, istemci kayıtlarında belirtilmeli ve yetkilendirme sunucusu bu eşleştirmeyi kullanarak hangi kimlik doğrulama şemasının uygulanacağını belirlemelidir.

3. Kimlik Doğrulama Yöntemi Belirlenmesi

- Yetkilendirme sunucusu, desteklediği kimlik doğrulama şemalarının her birinin güvenlik gereksinimlerini yerine getirdiğinden emin olmalıdır.
- Her kimlik doğrulama şeması, istemcinin güvenliğini sağlamak için **kimlik doğrulama ve yetkilendirme** süreçlerinde gereksinimlere uygun olmalıdır. Örneğin, bazı kimlik doğrulama yöntemleri, güvenli olmayan ağlarda kullanılmaya uygun olmayabilir ve bu nedenle sadece TLS (Transport Layer Security) ile korunmuş bağlantılarla sınırlı olmalıdır.

4. Eşleştirme Örneği

Bir istemci kaydında, istemci kimliği **client_id** belirtilir. Bununla birlikte, bu istemciye uygulanacak kimlik doğrulama şeması da tanımlanmalıdır. Örneğin:

- Eğer istemci, **OAuth 2.0 Client Assertion** kullanarak kimlik doğrulaması yapacaksa, yetkilendirme sunucusu bu istemcinin kaydını **Client Assertion** ile ilişkilendirebilir.
- Eğer istemci **Basic Authentication** kullanacaksa, bu kimlik doğrulama şeması da ilgili kayıtta belirtilir.

5. Özet

- **Yetkilendirme sunucusu**, **HTTP authentication** için yalnızca temel kimlik doğrulama şemaları değil, güvenlik gereksinimlerine uygun herhangi bir şemayı da destekleyebilir.
- Bu şemaların kullanılması durumunda, **client identifier** (istemci kimliği) ile ilgili kimlik doğrulama yöntemi arasında açık bir eşleştirme yapılmalıdır.
- Yetkilendirme sunucusunun, her kimlik doğrulama yönteminin güvenlik gereksinimlerini doğru şekilde yerine getirdiğinden emin olması gerekir.

Bu bölüm, OAuth 2.0 protokolünü daha esnek ve geniş bir uygulama yelpazesinde kullanılabilir kılmak için alternatif kimlik doğrulama yöntemlerine imkan tanır.

2.4. Unregistered Clients bölümü, OAuth 2.0 protokolünde, **kayıtsız istemciler** (unregistered clients) kullanımının mümkün olduğunu belirtmektedir, ancak bunun belirli kısıtlamaları ve ek gereksinimleri olduğu açıklanmaktadır. Şimdi bu kısmı detaylı olarak inceleyelim:

1. Kayıtsız İstemciler (Unregistered Clients)

- **Kayıtsız istemciler**, OAuth 2.0 protokolüne kaydedilmemiş istemcilerdir. Normalde OAuth 2.0'da, bir istemcinin çalışabilmesi için, istemcinin **yetkilendirme sunucusuna** kaydedilmesi gerekir. Ancak bu bölümde, OAuth 2.0 spesifikasyonu, kayıtsız istemcilerin de kullanılabileceğini kabul etmektedir.
- Kayıtsız istemciler, genellikle kimlik doğrulama veya yetkilendirme sürecine dahil olan, ancak **istemci kaydı** yapılmayan uygulamalardır. Bu, belirli durumlarda uygulamalara bazı esneklikler sağlayabilir, ancak protokolde bu kullanım için herhangi bir standart önerisi veya gereksinimi yoktur.

2. Kayıtsız İstemciler Kullanımının Kısıtlamaları

- **Kayıtsız istemcilerin kullanımı**, **bu spesifikasyonun kapsamı dışındadır**. Yani OAuth 2.0 protokolü, kayıtsız istemcilerin kullanımını desteklemez veya standart hale getirmez. Kayıtsız istemcilerin kullanımı, protokole dahil edilmediği için bunun nasıl yapılacağı, güvenlik gereksinimleri ve uygulama uyumluluğu konusunda **ek güvenlik analizleri** ve **değerlendirmeler** yapılması gerekmektedir.
- Kayıtsız istemcilerin kullanımı, **güvenlik riskleri** ve **interoperabilite** (uyumluluk) sorunlarına yol açabilir. Özellikle, istemci kimlik doğrulama sürecinde kaydın yapılmaması, **kimlik doğrulama ve yetkilendirme işlemlerinin** daha az güvenli olmasına sebep olabilir. Bu nedenle, kayıtsız istemcilerin kullanımı, dikkatli bir şekilde gözden geçirilmeli ve **ek güvenlik önlemleri** alınmalıdır.

3. Ek Güvenlik İncelemesi ve Uyumluluk

- Kayıtsız istemciler, OAuth 2.0 protokolünde yer almadığı için, güvenlik açısından **ilave analizler** gerektirir. Çünkü kayıtsız istemcilerin yetkilendirme süreci, protokolle uyumlu olmayabilir ve istemci ile yetkilendirme sunucusu arasındaki güven ilişkisi zayıflayabilir.
- Ayrıca, kayıtsız istemcilerin kullanımı, OAuth 2.0 spesifikasyonunun diğer taraflarıyla uyumsuzluk yaratabilir. Bu nedenle, kayıtsız istemcilerle ilgili uygulanabilirlik ve güvenlik değerlendirmelerinin yapılması gerekmektedir.

4. Özet

- **Kayıtsız istemciler** OAuth 2.0'da **resmi olarak desteklenmemektedir**, ancak kullanımları mümkündür.
- Kayıtsız istemcilerin kullanımı **güvenlik riskleri** ve **uyumluluk sorunları** oluşturabilir, bu nedenle dikkatli bir şekilde incelenmesi ve ek güvenlik önlemlerinin alınması gerekmektedir.
- Spesifikasyon, kayıtsız istemciler için özel bir düzenleme veya standart sunmadığı için, bu tür istemcilerin kullanımı, protokolle uyumsuzluk oluşturabilir.

Bu bölüm, **OAuth 2.0 protokolü** için güvenlik açısından belirli bir **koruma sağlamak** amacıyla istemcilerin kaydını **gereklilik** olarak belirlemiştir. Ancak kayıtsız istemciler için bazı durumlar dışında, belirli kullanım senaryolarında bu tür istemcilerin kullanılabileceği belirtiliyor.

3.1. Authorization Endpoint bölümü, OAuth 2.0 protokolünün **yetkilendirme uç noktası** (authorization endpoint) ile ilgili gereksinimleri açıklar. Yetkilendirme uç noktası, istemcinin kullanıcıdan (kaynak sahibi) **yetkilendirme izni** almak amacıyla kullandığı bir kaynaktır. Bu uç nokta, OAuth 2.0 protokolünün temel taşlarından biridir ve aşağıdaki önemli unsurları içerir:

1. Yetkilendirme Uç Noktasının Amacı

- **Yetkilendirme grant'ı alma:** Yetkilendirme uç noktası, istemcinin **yetkilendirme grant'ı** almak için kullanılır. Bu grant, istemcinin kaynağa (örneğin, API'ye) erişim izni almasını sağlar. Kullanıcı, yetkilendirme işlemi sırasında istemcinin belirttiği yetkilendirme uç noktasına yönlendirilir ve burada onay vermesi beklenir.

2. Kaynak Sahibinin Kimlik Doğrulaması

- **Kimlik doğrulama gereksinimi:** Yetkilendirme sunucusu, kaynak sahibinin kimliğini doğrulamalıdır. Kimlik doğrulama işlemi (örneğin, kullanıcı adı ve şifre girişi veya oturum çerezleri) bu spesifikasyonun dışında olsa da, bu adım **gereklidir**.
- Kimlik doğrulama işlemi, kullanıcının doğru olduğuna ve kaynak sahibinin izin verme yetkisine sahip olduğuna emin olmak için yapılır.

3. Yetkilendirme Uç Noktasının Konumu

- **Uç nokta konumunun belirlenmesi:** İstemci, yetkilendirme uç noktasının yerini öğrenmek için belirli yöntemler kullanır. Bu spesifikasyon, bu yöntemi belirtmez ancak genellikle uç nokta konumu, hizmetin dokümantasyonunda sağlanır.

4. Authorization Endpoint URI Yapısı

- **Query Parametreleri ve Formülasyonu:** Yetkilendirme uç noktasının URI'si, **"application/x-www-form-urlencoded"** formatında sorgu parametreleri içerebilir. Bu parametreler, RFC3986'ya uygun şekilde eklenmelidir ve parametrelerin sırası değiştirilmemelidir. Ancak **fragment** (örn. # işareti ile başlayan kısımlar) kullanımı yasaktır.
- **Tekrar eden parametreler:** Aynı parametre, istekte birden fazla kez gönderilemez. Eğer bir parametre eksikse, sunucu bu durumu, parametrenin yok sayılması olarak kabul etmelidir.

5. Güvenli İletişim Gereksinimi

- **TLS Kullanımı:** Yetkilendirme uç noktasına yapılan talepler, **kullanıcı kimlik bilgilerini** ve **açık metin verileri** ilettiğinden, bu verilerin güvenli bir şekilde iletilmesi önemlidir. Bu nedenle, **TLS (Transport Layer Security)** protokolü kullanımı **zorunludur**. Yani, HTTPS üzerinden iletişim yapılmalıdır.
- **Veri Güvenliği:** Bu gereklilik, kullanıcının kimlik bilgilerini korumak ve olası kötü niyetli saldırıları engellemek için gereklidir.

6. HTTP Yöntemleri: GET ve POST

- **GET Yöntemi:** Yetkilendirme uç noktası, en azından **HTTP GET** yöntemini desteklemelidir. Bu yöntem, genellikle URL üzerinden sorgu parametreleri gönderilmesini sağlar.

- **POST Yöntemi:** Yetkilendirme uç noktası **POST** yöntemini de destekleyebilir, ancak bu zorunlu değildir. **POST**, genellikle daha büyük veri miktarlarının gönderilmesi gerektiğinde tercih edilen bir yöntemdir.

7. Tanımadık Parametreler ve Değerler

- **Tanımlanmayan parametreler:** İstemciden gelen tanınmayan parametreler, yetkilendirme sunucusu tarafından **yok sayılmalıdır**. Sunucu bu parametreleri dikkate almaz.
- **Parametre Değerlerinin Yok Sayılması:** Parametrelerin değerleri eksikse, yetkilendirme sunucusu bu parametreleri **omiş** olarak kabul etmelidir.

8. Tekrar Eden Parametreler

- **Bir parametre birden fazla kez gönderilmemelidir:** Herhangi bir parametre birden fazla kez istemci tarafından gönderilmemelidir. Eğer böyle bir durum söz konusuysa, parametrelerin her birini tek seferde işlemesi gereklidir.

Özet:

- Yetkilendirme uç noktası, istemcinin kaynağa erişim izni almak için kullandığı ana kaynaktır.
- Sunucu, kullanıcı kimliğini doğrulamalıdır.
- Yetkilendirme uç noktasına yapılan talepler, **TLS** üzerinden güvenli bir şekilde iletilmelidir.
- İstemci, **GET** ve gerekirse **POST** yöntemlerini kullanabilir.
- **Query parametreleri**, belirli bir formata uygun şekilde iletilmeli, tekrar edilmemeli ve sunucu tanımadığı parametreleri göz ardı etmelidir.

Bu uç nokta, OAuth 2.0 protokolünde kullanıcının kimliğini doğrulamak ve ona erişim izni sağlamak amacıyla kritik bir rol oynar.

3.1.1. Response Type bölümü, **authorization endpoint** (yetkilendirme uç noktası) üzerinden gönderilen taleplerin nasıl yanıtlanacağına dair önemli bir parametreyi açıklar. Bu parametre, istemcinin hangi tür **yetkilendirme yanıtını** istediğini belirtir. İstemci, **authorization server** (yetkilendirme sunucusuna) bu parametreyi kullanarak hangi tür **grant** (izin) tipini talep ettiğini bildirir. Bu bölümde açıklanan "response_type" parametresi, OAuth 2.0 protokolündeki farklı yetkilendirme akışlarını belirler. Aşağıda **response_type** parametresinin nasıl çalıştığı ve olası değerleri açıklanmıştır:

1. response_type Parametresi

- **Zorunlu Parametre:** "response_type" parametresi, yetkilendirme isteği gönderilirken **zorunlu** bir parametredir. Bu parametre, istemcinin hangi tür **yetkilendirme yanıtını** beklediğini belirtir.
- İstemci, yetkilendirme sunucusuna hangi tür yanıt almak istediğini bildirir ve sunucu buna göre uygun bir işlem yapar.

2. response_type Değerleri

İstemcinin talep ettiği yanıt türüne göre, **response_type** parametresi farklı değerler alabilir. Bu değerler, istemcinin yetkilendirme sunucusundan alacağı **yetkilendirme grant'ı** türünü belirler.

a. code (Authorization Code Grant)

- **Açıklama:** Bu, **authorization code grant** türü için kullanılır. Bu, OAuth 2.0'ın en yaygın akışlarından biridir. İstemci, yetkilendirme sunucusundan bir **authorization code** (yetkilendirme kodu) almak ister. Bu kod, daha sonra istemci tarafından **token endpoint**'e gönderilir ve bir **access token** (erişim jetonu) alınır.
- **Ne Zaman Kullanılır?**: Bu yanıt türü, istemcinin güvenli bir şekilde yetkilendirme kodunu almasını isteyen istemciler için kullanılır. Bu tip genellikle web uygulamaları gibi **gizlilik** gereksinimi yüksek olan istemciler tarafından kullanılır.

b. token (Implicit Grant)

- **Açıklama:** Bu, **implicit grant** türü için kullanılır. İstemci doğrudan bir **access token** (erişim jetonu) almak ister, ancak authorization code alınmaz. Bu tür genellikle daha az güvenlik gereksinimi olan istemciler için uygundur (örneğin, tarayıcı tabanlı uygulamalar veya tek sayfalık uygulamalar - SPA).
- **Ne Zaman Kullanılır?**: Bu, istemcinin daha hızlı bir şekilde erişim sağlamak istediği, ancak daha düşük güvenlik önlemleri gerektiren uygulamalarda kullanılır.

c. Extension Values (Uzantı Değerleri)

- **Açıklama:** OAuth 2.0 protokolüne ek olarak bazı özel uzantılar (extension grant types) olabilir. Bu uzantı türleri, **response_type** parametresinde özel bir değer alabilirler. Bu uzantılar, OAuth 2.0'ın dışında olan ancak OAuth 2.0 uyumlu sistemlere özgü özellikler sunabilir.
- **Örnek:** "a b" gibi bir **response_type** değeri olabilir, burada **"a"** ve **"b"** değerleri uzantı yanıt türlerini temsil eder ve sıralama fark etmez. Örneğin, bir OAuth uzantısı, her iki türdeki yanıtları aynı anda almayı mümkün kılabilir.
- **Ne Zaman Kullanılır?**: Özel uygulama gereksinimleri veya ek OAuth 2.0 uzantıları kullanıldığında.

3. response_type Parametresinin Eksik Olması veya Anlaşılmaması Durumu

- **Eksik veya Tanınmayan response_type:** Eğer istemci, **response_type** parametresini göndermemişse veya gönderdiği değer yetkilendirme sunucusu tarafından anlaşılmıyorsa, yetkilendirme sunucusu bir **hata yanıtı** döndürmelidir. Bu hata yanıtı, **Section 4.1.2.1**'de açıklanan şekilde yapılandırılmalıdır.
    - **Hata Durumu:** Yetkilendirme sunucusu, beklenen "response_type" değerini alamazsa veya eksikse, bu durumda **400 Bad Request** gibi bir hata dönebilir ve istemciye uygun hata mesajını iletebilir.

4. Uzantı Yanıt Türlerinin (Extension Response Types) Kullanımı

- Uzantı yanıt türleri, **response_type** parametresine bir **boşluk ile ayrılmış bir dizi değer** ekleyebilir. Bu tür yanıtların anlamı, ilgili uzantı spesifikasyonları tarafından tanımlanır.

- **Örnek:** "a b" ve "b a" yanıt türleri eşdeğerdir; sıralama önemli değildir.

Özet:

- **response_type** parametresi, istemcinin yetkilendirme uç noktasına yaptığı isteklerde hangi tür yanıt beklediğini belirtir.
- Bu parametre, **"code"** (authorization code grant) veya **"token"** (implicit grant) gibi değerler alabilir.
- Uzantı yanıt türleri, belirli OAuth 2.0 uzantıları tarafından tanımlanmış özel değerler olabilir.
- Eğer **response_type** parametresi eksik ya da geçersizse, yetkilendirme sunucusu hata mesajı döndürmelidir.

Bu parametre, istemci ile yetkilendirme sunucusu arasındaki yetkilendirme akışının doğru şekilde başlatılmasını sağlar.

3.1.2. Redirection Endpoint bölümü, OAuth 2.0 protokolündeki bir **redirection endpoint** (yönlendirme uç noktası) kullanımıyla ilgilidir. Yönlendirme uç noktası, yetkilendirme süreci sonunda kullanıcının (kaynak sahibi) onayı alındığında istemciye geri dönmesini sağlar. İstemci, yetkilendirme sunucusuyla etkileşim tamamlandığında, kullanıcıyı bu uç noktaya yönlendirir. Bu bölümdeki açıklamalar, yönlendirme uç noktasının nasıl yapılandırılacağına ve nasıl çalışması gerektiğine dair ayrıntıları sunar.

1. Redirection Endpoint'ın Rolü

- Yetkilendirme süreci tamamlandıktan sonra, **authorization server** (yetkilendirme sunucusu), kaynak sahibinin **user-agent** (kullanıcı aracısı, yani tarayıcı) aracılığıyla istemciye geri yönlendirme yapar.
- **Yönlendirme uç noktası**, istemcinin, yetkilendirme sunucusuyla yapılan etkileşimden sonra alacağı yanıtı alacağı yerdir. Bu uç nokta, istemcinin kaydolduğu ve yetkilendirme isteği sırasında belirttiği yerdir.
- Yönlendirme uç noktası, istemciye dönüş için belirlenen bir **URI**'dir (Uniform Resource Identifier), ve bu URI, yetkilendirme isteği sırasında önceden belirlenmiş olmalıdır.

2. Redirection Endpoint URI

- **Mutlaka Tam URI Olmalı:** Yönlendirme uç noktası bir **absolute URI** olmalıdır. Bu, URI'nin başında **şema (örn. http://)**, **host** (sunucu adı) ve **path** (yol) gibi bileşenlerin yer alması gerektiği anlamına gelir.
    - **RFC 3986**'ya göre, bu bir tam URI olmalıdır ve bu RFC'de tanımlanan kurallara uymalıdır.
- **Query Bileşeni:** Yönlendirme uç noktası URI'si, bir **"application/x-www-form-urlencoded"** biçiminde query parametreleri içerebilir. Bu query parametreleri, istemcinin yetkilendirme sunucusundan dönen bilgileri taşır.
    - Bu query bileşeni, URI'ye ek parametreler eklenmeden önce tutulmalı, yani önceden var olan parametrelerin kaybolmaması sağlanmalıdır.
    - Parametreler **Appendix B**'de tanımlandığı şekilde şifrelenmiş formda olabilir.

- **Fragment İçermemeli:** Yönlendirme URI'si **fragment** bileşeni içeremez. **Fragment** kısmı, URI'nin parçası olarak sadece istemci tarafında işlenen bir bölümdür ve yetkilendirme sürecinde herhangi bir sunucu tarafı işlemi için kullanılmaz. Bu yüzden, OAuth 2.0'da yönlendirme URI'leri fragment içermemelidir.

3. Yönlendirme Endpoint'inin İşlevi ve Kısıtlamalar

- **İstemci Kaydı ve Yetkilendirme İsteği:** Yönlendirme uç noktası, istemcinin **client registration** (istemci kaydı) sırasında belirlediği bir URI'dir. Yetkilendirme isteği yapılırken, istemci bu URI'yi belirtir.
    - Örneğin, bir istemci kaydolurken, "http://client.example.com/callback" gibi bir yönlendirme URI'si belirlerse, yetkilendirme sunucusu bu URI'yi kullanarak kullanıcıyı geri gönderir.
- **Kullanıcı-Agent (Tarayıcı) ile Etkileşim:** Yönlendirme, kullanıcıyı tekrar istemciye yönlendirmek için kullanıcı aracısı (tarayıcı) üzerinden yapılır. Yetkilendirme sunucusu, istemciye gerekli erişim izinlerini verdikten sonra, kullanıcının tarayıcısını bu URI'ye yönlendirir.

Özet:

- **Redirection endpoint** (yönlendirme uç noktası), kaynak sahibinin (kullanıcının) yetkilendirme işlemini tamamladıktan sonra istemciye geri gönderileceği yerdir.
- Yönlendirme URI'si **mutlaka tam bir URI** olmalıdır ve **fragment** içeremez.
- Yönlendirme URI'sine eklenen query parametreler **"application/x-www-form-urlencoded"** biçiminde olmalı ve daha sonra ek parametreler eklenmeden önce korunmalıdır.
- Bu uç nokta, istemcinin kaydolduğu ve yetkilendirme sırasında belirttiği yerdir.

Bu bölüm, istemcilerin ve yetkilendirme sunucularının **OAuth 2.0 akışları** sırasında güvenli ve uyumlu bir şekilde nasıl iletişim kurması gerektiğini tanımlar.

3.1.2.1. Endpoint Request Confidentiality bölümü, OAuth 2.0 protokolünde **yönlendirme uç noktasına** yapılan isteklerin güvenliği ile ilgilidir. Bu bölüm, özellikle kullanıcıya ait hassas bilgilerin iletildiği durumlarda güvenlik önlemleri alınmasını vurgular.

1. TLS Kullanımı ve Öneri

- **TLS (Transport Layer Security)**, verilerin internet üzerinden güvenli bir şekilde iletilmesini sağlayan bir şifreleme protokolüdür. Bu bölümde, **TLS**'nin kullanımı teşvik edilir.
    - **TLS Kullanımı Gereklidir:** Eğer istemci, **"code"** (yetkilendirme kodu) veya **"token"** (erişim belirteci) gibi yanıt türlerini talep ediyorsa veya yönlendirme isteği, **açık ağda hassas kimlik bilgileri** iletilecekse, **TLS** kullanılması **ŞARTTIR**.
    - **Hassas Bilgilerle İletişim:** Kullanıcı adı, şifre, yetkilendirme kodu veya erişim belirteci gibi hassas veriler açık ağlarda iletildiğinde, bu verilerin **gizliliği ve bütünlüğü** bozulabilir. TLS bu verileri şifreleyerek güvenliği artırır.

2. TLS Zorunluluğunun Uygulanmaması

- Bu spesifikasyon, **TLS'nin zorunlu hale getirilmesini** istememektedir çünkü o dönemde **TLS'i uygulamak** çoğu istemci geliştiricisi için büyük bir engel olabilir. Yani, bazı istemcilerin TLS desteği olmayabilir veya kullanımı zorlu olabilir.
    - Ancak, **TLS'nin sağlanamadığı durumlarda**, yetkilendirme sunucusu, kaynak sahibine (kullanıcıya) **güvensiz uç nokta** hakkında bir uyarı mesajı göstermelidir. Bu, kullanıcıyı güvenlik riski hakkında bilgilendirmek için yapılır.

      Örneğin, kullanıcı bir üçüncü parti giriş hizmeti (third-party sign-in) kullanıyorsa, yetkilendirme sunucusu kullanıcıya şu uyarıyı gösterebilir: "Bu bağlantı güvenli değil. Kimlik bilgilerinizi risk altında bırakabilirsiniz."

3. TLS Kullanımının Güvenlik Üzerindeki Etkisi

- **TLS'in eksikliği, istemci ve korunan kaynaklar üzerinde ciddi güvenlik etkileri yaratabilir.** Yönlendirme uç noktasındaki isteklerin açık ağda iletilmesi, hassas verilerin saldırganlar tarafından ele geçirilmesine, değiştirilmesine veya kötüye kullanılmasına neden olabilir. Bu, OAuth 2.0'ın temel güvenlik garantilerinden biri olan **kimlik doğrulama** ve **yetkilendirme** süreçlerini zayıflatabilir.
- Özellikle OAuth 2.0 kullanılarak gerçekleştirilen **üçüncü taraf kimlik doğrulaması** (third-party sign-in) gibi durumlarda, **TLS kullanımı kritik** hale gelir. Çünkü bu senaryolarda, kullanıcı oturumu genellikle bir üçüncü parti uygulama tarafından devralınır ve kimlik bilgileri (örneğin, sosyal medya hesapları veya banka hesapları) çok daha hassas hale gelir.

4. Özet

- **TLS kullanımı önerilir** ve genellikle "code" veya "token" yanıt türleri talep edildiğinde, veya hassas bilgilerin iletilmesi bekleniyorsa zorunludur.
- Eğer **TLS sağlanamıyorsa**, yetkilendirme sunucusu kullanıcıyı **güvensiz uç nokta hakkında uyar** yapmalıdır.
- **Güvenlik** açısından, TLS, özellikle **üçüncü parti kimlik doğrulaması** gibi durumlarda büyük önem taşır. Bu güvenlik protokolü, **hassas bilgilerin korunması** ve **veri iletiminin gizliliği** için gereklidir.

Bu bölüm, güvenli veri iletiminin önemini vurgular ve yetkilendirme sürecinin güvenli bir şekilde yapılması için TLS kullanılması gerektiğini belirtir.

3.1.2.2. Registration Requirements bölümü, **OAuth 2.0**'da **yönlendirme uç noktası (redirection endpoint)** kayıt sürecini tanımlar. Bu kayıt, güvenliği sağlamak ve potansiyel saldırılara karşı korunmak için gereklidir. Yönlendirme uç noktası, kullanıcının yetkilendirme işlemi tamamlandığında OAuth sunucusu tarafından geri gönderileceği URI'dir. Bu bölümde, yönlendirme uç noktalarının nasıl kaydedileceği ve hangi tür istemcilerin kaydettirilmesi gerektiği açıklanır.

1. Kayıt Zorunluluğu Olan İstemciler

Yetkilendirme sunucusu, aşağıdaki istemcilerin **yönlendirme uç noktasını** kaydettirmelerini **zorunlu kılmalıdır**:

- **Public clients (Açık istemciler)**: Bu istemciler, istemci kimlik bilgilerini gizli tutamazlar, yani uygulama kodu kullanıcının cihazında çalışır ve kimlik bilgileri korunmaz. Bu nedenle, bu istemcilerin yönlendirme uç noktalarının kaydedilmesi gereklidir.
- **Confidential clients utilizing the implicit grant type (Yetkilendirme kodu kullanılmayan gizli istemciler)**: İstemci, güvenli kimlik doğrulama yapabilen, fakat açık istemciler gibi kimlik bilgilerini riske atan istemcilere kıyasla daha güvenlidir. Ancak, **implict grant type** kullanarak erişim belirteci elde etmeyi amaçlayan istemciler için de yönlendirme uç noktası kaydı gereklidir.

2. Tüm İstemcilerin Kayıt Zorunluluğu

- Yetkilendirme sunucusu, **tüm istemcilerin** yönlendirme uç noktasını, **yetkilendirme uç noktasını** kullanmadan önce kaydetmelerini **önerir**. Bu, yalnızca güvenli istemciler için değil, tüm istemciler için önemlidir.
- Yetkilendirme sunucusunun, istemcilerden kaydetmeleri gereken yönlendirme URI'yi **tam** olarak talep etmesi gerekir. Yani istemci sadece bir kısmı değil, URI'nin tamamını kaydetmelidir. Ancak, bu zorunlu değilse, sunucu, istemcinin yalnızca **URI şeması**, **yetki kısmı** ve **yol kısmı** (path) gibi temel bileşenleri kaydetmesini isteyebilir. Bu durumda istemci, sorgu bileşenini (query component) dinamik olarak değiştirebilir.

3. Birden Fazla Yönlendirme Uç Noktasının Kaydı

- Yetkilendirme sunucusu, istemcinin birden fazla yönlendirme uç noktası kaydetmesine **izin verebilir**. Bu, istemcinin birden fazla yetkilendirme uç noktası kullanabileceği anlamına gelir.

4. Yönlendirme Uç Noktası Kaydının Olmaması Riskleri

- Eğer yönlendirme URI'sinin kaydedilmesi zorunlu kılınmazsa, kötü niyetli bir saldırgan **açık yönlendirme** (open redirector) saldırısı yapabilir. Bu tür saldırılar, yetkilendirme uç noktasının **başka bir siteye yönlendirme yapacak şekilde manipüle edilmesi** ile gerçekleştirilir. Bu saldırılar, yetkilendirme sunucusunun istemcinin doğruluğunu doğru şekilde kontrol etmemesinden faydalanır.

  Örneğin:

  - **Açık yönlendirme** durumu şu şekilde gerçekleşebilir: Eğer istemci tarafından kayıtlı olmayan bir yönlendirme uç noktası, kötü niyetli bir siteye yönlendiriliyorsa, kullanıcı oturum açtıktan sonra, kimlik bilgileri yanlış bir siteye yönlendirilebilir ve saldırgan bu bilgileri ele geçirebilir.

5. Özetle

- **Public** ve **Confidential (implicit grant)** istemcilerinin yönlendirme uç noktalarını kaydetmeleri gereklidir.
- Tüm istemcilerin yönlendirme uç noktalarını kaydetmesi önerilir.
- Yönlendirme URI'sinin **tam** olarak kaydedilmesi önemlidir; yalnızca şema, yetki ve yol kısmı kaydedilerek, istemci sorgu parametrelerini dinamik olarak değiştirebilir.
- Yetkilendirme sunucusu, birden fazla yönlendirme uç noktasının kaydedilmesine izin verebilir.

- **Açık yönlendirme** (open redirector) saldırılarından korunmak için yönlendirme URI'sinin kaydedilmesi çok önemlidir.

Bu bölüm, istemci güvenliğini sağlamak ve kötüye kullanımı engellemek için doğru yönlendirme uç noktası kaydının yapılmasının önemini vurgular.

3.1.2.3. Dynamic Configuration bölümü, **OAuth 2.0** protokolünde istemcilerin yönlendirme URI'leri (redirection URIs) ile nasıl çalıştığını ve yetkilendirme sunucusunun bu URI'lerle nasıl etkileşimde bulunduğunu açıklamaktadır. Bu bölüm, istemcilerin yönlendirme URI'lerini dinamik bir şekilde kullanabilmelerini sağlar ve ayrıca, istemcinin kayıtlı URI'ler ile uyumsuzluk durumunda nasıl bir işlem yapılacağını belirtir.

1. Yönlendirme URI'lerinin Kayıt Durumu

- Eğer istemci birden fazla yönlendirme URI'si kaydettirmişse veya yalnızca URI'nin bir kısmı (örneğin, şema, yetki ve yol) kaydedildiyse, istemci yetkilendirme isteğinde bulunduğunda **yönlendirme URI'sini "redirect_uri"** parametresiyle sunucuya göndermelidir. Bu, istemcinin tam yönlendirme URI'sini belirttiği ve yetkilendirme sunucusunun bu URI'yi doğrulaması gerektiği anlamına gelir.
- Eğer istemci **hiçbir yönlendirme URI'si kaydettirmemişse**, yine de **"redirect_uri"** parametresi ile yönlendirme URI'sini belirtmek zorundadır.

2. Yönlendirme URI'sinin Doğrulama Süreci

- **Yönlendirme URI'si** yetkilendirme isteğiyle birlikte sunucuya gönderildiğinde, yetkilendirme sunucusu şu adımları izler:
    1. **Kaydedilen URI'lerle Karşılaştırma**: Yetkilendirme sunucusu, istemcinin göndermiş olduğu yönlendirme URI'sini, kaydedilen yönlendirme URI'leri (veya URI bileşenleri) ile karşılaştırmalıdır. Bu karşılaştırma, **RFC 3986** bölüm 6'ya uygun olarak yapılmalıdır.
    2. **Tam URI Karşılaştırması**: Eğer istemci kaydında **tam yönlendirme URI'si** verilmişse, sunucu bu URI'yi doğrudan karşılaştırmalıdır. Bu karşılaştırma, basit bir **string (dize) karşılaştırması** olarak yapılır. Yani, istemcinin belirttiği URI ile kaydedilen URI'nin tam olarak eşleşip eşleşmediği kontrol edilir.
        - Örneğin, istemcinin kaydettiği yönlendirme URI'si `https://example.com/callback` ise, yetkilendirme sunucusu, istemcinin gönderdiği `redirect_uri` parametresinin bu tam URI ile eşleşip eşleşmediğini kontrol eder.

3. Yönlendirme URI'si Eşleşmeme Durumu

- Eğer istemcinin gönderdiği yönlendirme URI'si, kaydedilen URI'ler ile eşleşmezse, yetkilendirme sunucusu bu durumu hata olarak değerlendirmeli ve uygun bir hata yanıtı döndürmelidir. Bu genellikle, istemcinin kaydetmediği bir URI'ye yönlendirme yapmaya çalışması durumunda gerçekleşir.

4. Esneklik ve Dinamik Yönlendirme

- **Dinamik yapılandırma** sayesinde istemciler, kaydedilen URI'leriyle uyumsuzluk durumunda bile, belirli URI bileşenlerini dinamik olarak değiştirebilirler. Örneğin, istemci sadece URI'nin temel kısmını (şema, yetki, yol) kaydedebilir ve sorgu parametrelerini (query components) dinamik olarak değiştirebilir. Bu esneklik, istemcinin yönlendirme URI'sini daha esnek bir şekilde yönetmesine olanak tanır.

5. Özetle

- Eğer istemci birden fazla veya eksik yönlendirme URI'si kaydettirmişse, istemci her yetkilendirme isteğiyle birlikte tam yönlendirme URI'sini **"redirect_uri"** parametresi olarak göndermelidir.
- Yetkilendirme sunucusu, gönderilen yönlendirme URI'sini kaydedilen URI ile karşılaştırmalıdır. Bu karşılaştırma basit bir string karşılaştırmasıdır ve URI'nin tam olarak eşleşip eşleşmediği kontrol edilir.
- Bu süreç, istemci tarafından kullanılan URI'yi doğrulamak için gereklidir ve güvenlik amacıyla doğru eşleşme yapılmalıdır.

Bu özellik, istemcilerin dinamik yapılandırmalarını destekler ve yalnızca güvenli ve kayıtlı URI'lerin yönlendirme için kullanılmasına olanak tanır.

3.1.2.4. Invalid Endpoint bölümü, **OAuth 2.0** protokolünde, istemciden gelen geçersiz yönlendirme URI'leri (redirection URIs) ile ilgili olarak yetkilendirme sunucusunun nasıl davranması gerektiğini açıklamaktadır.

1. Geçersiz Yönlendirme URI'si Durumu

Eğer bir yetkilendirme isteği şu nedenlerden dolayı başarısız olursa:

- **Eksik yönlendirme URI** (yönlendirme URI'si parametresi gönderilmemişse),
- **Geçersiz yönlendirme URI** (geçersiz bir URI formatı veya hatalı yapılandırılmış bir URI gönderilmişse),
- **Uyumsuz yönlendirme URI** (gönderilen URI, kayıtlı URI ile eşleşmiyorsa),

o zaman **yetkilendirme sunucusu** şu şekilde hareket etmelidir:

2. Bilgilendirme ve Kullanıcı Yönlendirmesi

- **Yetkilendirme sunucusu**, **kaynak sahibi** (resource owner) olarak bilinen kullanıcının, yanlış bir yönlendirme URI'sine yönlendirilmeden önce bu hatayı bilgilendirmelidir. Yani, kullanıcıya bir hata mesajı gösterilmeli ve hatanın ne olduğunu anlaması sağlanmalıdır.
    - Örneğin, bir kullanıcı bir yetkilendirme işlemi yaparken, sistem geçersiz bir URI'yi doğrulamaya çalıştığında, sistem kullanıcıyı "Geçersiz yönlendirme URI" hatası ile bilgilendirebilir.
- **Otomatik Yönlendirme Yasaktır**: Yetkilendirme sunucusu, geçersiz bir yönlendirme URI'sine otomatik olarak kullanıcıyı yönlendirmemelidir. Yani, eğer bir hata oluşursa, kullanıcı doğrudan hatalı URI'ye yönlendirilmemelidir. Bunun yerine, sunucu bu hatayı bildirmeli ve kullanıcının doğru işlemi yapmasına olanak sağlamalıdır.

- Bu, güvenlik açısından kritik bir adımdır çünkü kötü niyetli bir kişi, geçersiz bir URI kullanarak kullanıcıyı zararlı bir siteye yönlendirmeyi amaçlayabilir. Yetkilendirme sunucusunun otomatik olarak hatalı URI'lere yönlendirme yapmaması, bu tür saldırılara karşı bir koruma sağlar.

3. Özetle

- **Geçersiz bir yönlendirme URI** geldiğinde, yetkilendirme sunucusu kullanıcılara bu hatayı bildirmelidir.
- Kullanıcı, hatalı URI'ye yönlendirilmemeli, doğru yönlendirme işlemi için hata mesajı gösterilmelidir.
- Bu davranış, güvenlik önlemleriyle uyumlu olup kötü niyetli saldırılara karşı bir koruma sağlar.

Bu bölüm, OAuth protokolünde güvenlik sağlamak ve kullanıcıyı olası hatalı yönlendirmelerden korumak için önemlidir.

3.1.2.5. Endpoint Content bölümü, OAuth 2.0 protokolünde, istemciye yönlendirme yapılan endpoint'in içerik güvenliği ile ilgili önlemleri tartışmaktadır. Bu bölüm, istemcilerin güvenli bir şekilde yönlendirme URI'lerinden alınan kimlik bilgilerini işlemeleri ve kullanıcı verilerini korumaları gerektiğini belirtir.

1. Yönlendirme Sonucu HTML Belgesi

Yönlendirme URI'sine yapılan bir istek genellikle bir **HTML belgesi** yanıtı döndürür. Bu HTML yanıtı, kullanıcının tarayıcısı (user-agent) tarafından işlenir ve gösterilir. Ancak, bu işlem sırasında **HTML belgesine eklenen scriptler**, URI'den alınan kimlik bilgilerine tam erişim sağlar. Bu da, kötü niyetli scriptlerin bu verilere müdahale edebilmesine veya onları dışarıya sızdırmasına olanak tanıyabilir.

Örneğin, bir istemci yönlendirme URI'sini içerik olarak gönderirse ve bu URI kimlik bilgileri içeriyorsa, içeriği işleyen tarayıcıdaki bir JavaScript kodu, bu kimlik bilgilerine erişebilir. Bu, özellikle kötü amaçlı scriptler için güvenlik açığı yaratabilir.

2. Üçüncü Taraf Scriptlerinin Kullanılmaması

- **İstemci, yönlendirme endpoint'inin yanıtında üçüncü taraf scriptlerini (örneğin, analitik araçları, sosyal medya eklentileri, reklam ağları) kullanmamalıdır.**
    - Üçüncü taraf scriptleri, kötü amaçlı yazılımlar, izleme scriptleri veya güvenlik açıkları taşıyabilir. Bu nedenle, istemci bu tür scriptleri kendi redirection endpoint'lerinde kullanmamalıdır.
- **Kimlik bilgilerini URI'den çıkartarak başka bir endpoint'e yeniden yönlendirme yapılmalıdır.**
    - Kimlik bilgileri URI'den çıkarılmalı ve güvenli bir şekilde başka bir endpoint'e yönlendirilmelidir. Bu, URI içinde hassas bilgilerin görünür olmamasını sağlar ve güvenliği artırır. Bu sayede, credential'lar (kimlik bilgileri) istemci tarafında herhangi bir şekilde sızmaz.

3. Scriptlerin Çalıştırılma Sırası

- Eğer istemci **üçüncü taraf scriptlerini kullanmak zorundaysa**, istemci, **kendi scriptlerinin** (kimlik bilgilerini URI'den çıkartıp güvenli bir şekilde işleyen scriptlerin) **öncelikli olarak çalışmasını sağlamalıdır.**
    - Bu, istemcinin kendi güvenlik önlemlerini alabilmesi için önemlidir. Kimlik bilgilerini URI'den çıkarıp başka bir endpoint'e yönlendirmeden önce, üçüncü taraf scriptlerinin bu verilere erişmesini engellemek gereklidir.

4. Özetle

- **Yönlendirme URI'si ile birlikte gelen HTML yanıtında** kimlik bilgileri yer alıyorsa, **üçüncü taraf scriptlerinin** bulunmaması gerekir.
- İstemci, **kimlik bilgilerini URI'den çıkartarak başka bir endpoint'e güvenli bir şekilde yönlendirme yapmalıdır.**
- **Üçüncü taraf scriptleri** kullanılacaksa, istemci, **kendi güvenlik scriptlerinin** öncelikli olarak çalışmasını sağlamalıdır.

Bu bölüm, güvenli olmayan uygulamalara karşı bir koruma sağlar ve istemcilerin kullanıcı kimlik bilgilerini üçüncü taraflardan gizli tutmalarını ve doğru şekilde işlemesini sağlar. Bu, OAuth protokolünün güvenli bir şekilde uygulanması açısından kritik bir güvenlik önlemidir.

3.2. Token Endpoint bölümü, OAuth 2.0 protokolündeki token endpoint'inin kullanımını ve güvenlik gereksinimlerini açıklar. Token endpoint, istemcinin **yetkilendirme belgesi** veya **yenileme token'ı** ile erişim token'ı almak için kullandığı bir kaynaktır. Bu süreç, kullanıcıya verilen yetkilendirme bilgileriyle istemcinin bir erişim token'ı almasını sağlar.

1. Token Endpoint Kullanımı

Token endpoint, istemcinin **yetkilendirme belgesini** veya **yenileme token'ını** sunarak bir **erişim token'ı** almasına olanak tanır. Bu endpoint, genellikle her yetkilendirme akışında kullanılır, ancak **implicit grant** akışında doğrudan erişim token'ı verildiği için bu endpoint kullanılmaz.

İstemcinin token endpoint'in yerini nasıl alacağı, bu spesifikasyonun kapsamı dışındadır, ancak genellikle **hizmetin belgelerinde** bu bilgi sağlanır.

2. Token Endpoint URI

Token endpoint URI'si, **application/x-www-form-urlencoded** biçiminde bir **sorgu parametresi** içerebilir. Bu parametreler, ek parametreler eklenirken **korunmalı** ve **URI'ye** eklenmelidir. Ancak, endpoint URI'si **fragment (parça) bileşeni** içermemelidir.

3. TLS Zorunluluğu

Token endpoint'e yapılan istekler, **kimlik bilgilerini** açık metin (clear-text) olarak taşıdığından, bu endpoint'e yapılan isteklerde **TLS (Transport Layer Security)** kullanılmalıdır. Bu, tüm erişim token'larının güvenli bir şekilde iletilmesini sağlamak için gereklidir. Bu, **SSL/TLS** şifrelemesi ile verilerin güvenli bir şekilde taşınmasını garanti eder.

4. HTTP Yöntemi

İstemci, token endpoint'e **erişim token'ı isteği** yaparken **POST** yöntemini kullanmalıdır. **GET** yöntemi burada kullanılmaz çünkü bu tür istekler genellikle kimlik doğrulama bilgilerini taşımaz ve daha az güvenlidir.

5. Parametreler ve Geçerlilik

- **Değeri olmayan parametreler**, istekten **ihmal edilmiş** gibi muamele görmelidir.
- **Tanımadığı parametreleri** yetkilendirme sunucusu **yok saymalıdır**.
- **Tekrar eden parametreler**, hem istek hem de yanıtlar için **tekrar edilmeyecek** şekilde işlenmelidir.

6. Özetle

- Token endpoint, istemcinin **erişim token'ı** almak için kullandığı **POST** tabanlı bir endpoint'tir.
- Bu endpoint'e yapılan isteklerin, kimlik bilgilerini güvenli bir şekilde taşıması için **TLS** kullanılmalıdır.
- İstek parametreleri, geçersiz veya eksik olduklarında doğru şekilde işlenmeli, ve tanınmayan parametreler görmezden gelinmelidir.

Bu bölümü anlamak, istemcinin güvenli bir şekilde erişim token'ları almasını ve istemci ile yetkilendirme sunucusu arasındaki iletişimin güvenliğini sağlamak açısından kritik öneme sahiptir.

3.2.1. Client Authentication bölümü, OAuth 2.0 protokolü çerçevesinde istemcilerin **token endpoint**'ine istek yaparken nasıl kimlik doğrulaması yapması gerektiğini açıklar. Özellikle, **confidential client'lar** (gizli istemciler) ve **client credentials** verilen diğer istemciler için kimlik doğrulama sürecinin nasıl işlemesi gerektiğini ele alır.

1. Kimlik Doğrulamanın Amacı

İstemci kimlik doğrulaması, aşağıdaki kritik amaçlar için gereklidir:

- **Yenileme token'ları ve yetkilendirme kodlarının istemciye bağlanmasını sağlamak**: Özellikle yetkilendirme kodları, güvenli olmayan bir kanal üzerinden **redirection endpoint**'ine gönderildiğinde veya redirection URI tam olarak kaydedilmediğinde kimlik doğrulaması kritik olur. Bu sayede, yalnızca belirli bir istemci, ona ait olan yetkilendirme kodlarını kullanabilir.
- **Tehlikeye düşmüş istemcilerin kurtarılması**: Eğer bir istemci tehlikeye düşerse, kimlik doğrulama sistemi kullanılarak istemcinin **kimlik bilgileri** değiştirilebilir ve **yeniden yapılandırılabilir**. Bu, saldırganların çalınan yenileme token'larını kullanarak erişim sağlamalarını engeller. Bu işlem, tüm yenileme token'larının iptal edilmesinden çok daha hızlı bir çözüm sağlar.
- **Kimlik doğrulama yönetimi ve güvenlik uygulamalarına uygunluk**: Kimlik doğrulama bilgilerini düzenli olarak güncellemek, bir güvenlik en iyi pratiğidir. İstemci kimlik bilgilerini döndürmek ve değiştirmek, yenileme token'larının tüm setlerini döndürmekten daha kolaydır. Bu da güvenlik açısından önemli bir avantaj sağlar.

2. Client ID Kullanımı

İstemci, token endpoint'ine istek gönderirken, kimliğini belirlemek için **client_id** parametresini kullanabilir. Bu parametre, istemcinin kimliğini tanımlamak için gereklidir ve isteğin doğru istemciye ait olduğunun doğrulanmasına yardımcı olur. Özellikle **authorization_code grant_type** kullanıldığında, istemci kimliği **client_id** parametresi olarak gönderilmelidir. Bu, istemcinin yanlışlıkla başka bir istemci için gönderilmiş olan yetkilendirme kodunu kabul etmesini engeller.

3. Kimlik Doğrulamanın Sağladığı Güvenlik

- **Yetkilendirme kodunun değiştirilmesi**: Client ID kullanmak, istemcinin yalnızca kendisi için verilmiş olan yetkilendirme kodunu kabul etmesini sağlar. Bu, **yetkilendirme kodu değiştirildiğinde** ya da **başka bir istemci** kodu kullanmaya çalıştığında, istemcinin bu durumu fark etmesine ve işlemi reddetmesine olanak tanır.
- **Korunan kaynaklar için ek güvenlik sağlamaz**: Ancak bu kimlik doğrulama, **korunan kaynaklara** ek güvenlik sağlamaz. Yalnızca istemcinin kendine ait olmayan bir yetkilendirme kodunu kullanmasını engeller.

4. Kimlik Bilgisi Değişiklikleri ve Güvenlik

İstemci kimlik bilgilerini düzenli olarak değiştirmek, güvenlik açısından önemlidir. Bu, **yenileme token'larını** kullanarak yapılan saldırıların önlenmesinde önemli bir adım olabilir. İstemci kimlik doğrulamasının bu şekilde yapılandırılması, uygulamanın uzun vadede güvenliğini artırır ve dış saldırılara karşı daha dayanıklı olmasını sağlar.

Özetle

- **Confidential clients** ve **client credentials** verilen diğer istemciler, **token endpoint**'ine yapılan isteklerde kimlik doğrulaması yapmalıdır.
- Kimlik doğrulamanın amacı, istemciye ait olan yetkilendirme kodlarının doğru istemci tarafından kullanılmasını sağlamak, tehlikeye düşmüş istemcileri kurtarmak ve kimlik doğrulama bilgilerini düzenli olarak güncellemektir.
- **client_id** parametresi, istemcinin doğru yetkilendirme kodunu almasını sağlamalıdır.
- Kimlik doğrulama, **korunan kaynaklara ek güvenlik sağlamaz**, ancak istemcinin doğru kaynaklara erişmesini garanti eder.

Bu süreç, istemcinin güvenli bir şekilde token almasını ve token'ların yanlış istemciler tarafından kullanılmasının önüne geçilmesini sağlar.

3.3. Access Token Scope bölümü, OAuth 2.0 protokolünde istemcilerin erişim taleplerini belirli bir **scope** (kapsam) ile sınırlandırmalarını ve bu kapsamın nasıl iletildiğini açıklar. Kapsam, istemcinin erişebileceği kaynaklar ve bu kaynaklar üzerinde hangi işlemleri gerçekleştirebileceği konusunda bir kısıtlama sağlar. Bu bölümde belirtilenler, istemci ile yetkilendirme sunucusu arasındaki etkileşimde **scope** parametresinin nasıl kullanılacağını anlatır.

1. Scope Parametresi

- **Scope** parametresi, istemcinin erişmek istediği kaynakları ve izinleri belirtmek için kullanılır. Bu parametre, istemcinin authorization (yetkilendirme) ve token (token alma) endpoint'lerine yaptığı isteklerde yer alabilir.
- **Scope** parametresi, bir dizi **boşlukla ayrılmış (space-delimited)** ve **büyük/küçük harfe duyarlı** kelimelerden oluşur. Bu kelimeler, yetkilendirme sunucusu tarafından tanımlanır.
- Birden fazla **scope-token** (kapsam belirleyicisi) içeren bir scope, her bir kelimenin erişim yetkisini artırdığı anlamına gelir. Yani birden fazla kelime (örneğin, "read write") belirtildiğinde, istemci her iki erişim hakkını talep eder.

  Örnek: `scope=read write` Bu örnekte, istemci hem okuma (read) hem de yazma (write) yetkisi talep etmektedir.

  Kapsamın Tanımı:

  *scope = scope-token *(SP scope-token)*
  scope-token = 1(%x21 / %x23-5B / %x5D-7E)*

Bu tanım, `scope-token`'ın hangi karakterleri içerebileceğini belirtir. Örneğin, `%x21` ASCII kodu ile "!" karakterini ifade eder.

2. Yetkilendirme Sunucusunun Cevabı

- Yetkilendirme sunucusu, istemciden gelen **scope** parametresini işleyerek, istemcinin talep ettiği **scope**'u doğrular. Ancak, yetkilendirme sunucusu bu **scope**'u tamamen veya kısmen göz ardı edebilir. Bunun nedeni, yetkilendirme sunucusunun güvenlik politikaları veya kaynak sahibinin (resource owner) verdiği talimatlar olabilir.
- Eğer yetkilendirme sunucusu, istemcinin talep ettiği **scope**'a uymayan bir **access token** verir ve verilen **scope** ile istemcinin talep ettiği **scope** farklıysa, **scope** yanıt parametresi kullanılarak istemciye verilen kapsam bildirilmelidir. Bu, istemcinin hangi erişim alanlarının sağlandığını anlamasını sağlar.

  Örnek: Eğer istemci `scope=read write` talep ettiyse ve sunucu yalnızca `read` izni veriyorsa, sunucu yanıtında şu şekilde bir **scope** parametresi dönebilir: `scope=read`.

3. Scope Parametresi Olmazsa Ne Olur?

- Eğer istemci, **scope** parametresini yetkilendirme talebinde belirtmezse, yetkilendirme sunucusu, bu durumda **önceden tanımlanmış bir varsayılan değer** ile isteği işleme alabilir. Eğer varsayılan bir değer yoksa, **invalid_scope** hatası dönebilir. Bu durumda istemci, hatanın nedeni olarak geçersiz bir **scope** olduğunu anlamalıdır.

- Yetkilendirme sunucusu, **scope** parametresinin ne olduğunu ve varsa varsayılan değeri **belgelerinde** açıklamalıdır.

4. Özet ve Güvenlik

- **Scope** parametresi, istemcinin erişmek istediği kaynakları sınırlamak için kullanılır. Bu, istemcinin sadece gerekli kaynaklara erişmesini ve gereksiz yetkileri talep etmemesini sağlar.
- Yetkilendirme sunucusu, istemcinin talep ettiği kapsamı tamamen ya da kısmen değiştirebilir, ancak istemciye bu durum bildirilmelidir.
- Eğer istemci **scope** parametresi göndermezse, yetkilendirme sunucusu varsayılan bir değer kullanabilir veya hatayı bildirebilir.

Bu mekanizma, OAuth 2.0 protokolünün daha esnek ve güvenli bir şekilde çalışmasını sağlar ve istemcilerin sadece gerekli erişim haklarını talep etmelerini teşvik eder.

4. Obtaining Authorization bölümü, OAuth 2.0 protokolünde istemcinin erişim token'ı almak için nasıl bir yetkilendirme (authorization) süreci başlatması gerektiğini ve bu sürecin nasıl işlediğini açıklar. Temelde, istemci erişim token'ı almak için **resource owner**'dan (kaynak sahibi) yetki alır. Bu yetkilendirme, istemcinin bir **authorization grant** (yetkilendirme izni) elde etmesine olanak tanır. İstemci, bu yetkilendirme iznini kullanarak erişim token'ını almak için token endpoint'ine başvurur.

1. Authorization Grant (Yetkilendirme İzni)

OAuth 2.0'da, bir istemcinin erişim token'ı talep etmesi için öncelikle **authorization grant** adı verilen bir izne ihtiyacı vardır. Bu **grant**, istemcinin kaynak sahibinden aldığı yetkiyi temsil eder. Authorization grant, istemcinin belirli bir kaynağa veya kaynağa dair belirli izinlere erişim hakkı kazandığını gösterir.

Authorization grant türleri şunlardır:

2. Grant Types (Yetkilendirme Türleri)

OAuth 2.0, istemcilerin yetki alırken kullanabileceği dört ana yetkilendirme türü tanımlar. Her tür, farklı kullanım senaryolarına hizmet eder ve güvenlik gereksinimlerine göre farklı şekillerde işlenir:

- **Authorization Code Grant (Yetkilendirme Kodu Türü)**: Bu tür, istemciye kullanıcı tarafından bir yetkilendirme kodu verilmesini gerektirir. Genellikle sunucu tarafı uygulamaları için kullanılır. İstemci, kullanıcıyı yetkilendirme sunucusuna yönlendirir, kullanıcı orada kimliğini doğrular ve onay verir. Sonrasında, yetkilendirme sunucusu istemciye bir kod (authorization code) döner. Bu kod, istemcinin **token endpoint**'inden bir erişim token'ı almak için kullanılır. Bu tür, yüksek güvenlik sağlar çünkü client secret (istemci kimliği) istemcinin sunucusunda saklanır ve erişim token'ı sunucudan alınır.
- **Implicit Grant (Açık Erişim Türü)**: Bu tür, istemcinin doğrudan bir erişim token'ı almasını sağlar. Genellikle tarayıcı tabanlı uygulamalar (client-side) için kullanılır. İstemci, yetkilendirme sunucusuna kullanıcıyı yönlendirir ve kullanıcı kimliğini doğruladıktan sonra erişim token'ı doğrudan istemciye yönlendirilir. Implicit grant genellikle daha düşük güvenlikli ve hızlıdır, ancak token'lar doğrudan istemcinin tarayıcısında bulunur, bu da güvenlik risklerini artırabilir.
- **Resource Owner Password Credentials Grant (Kaynak Sahibi Şifre Kimlik Doğrulaması Türü)**: Bu türde, kullanıcı, kullanıcı adı ve şifresini doğrudan istemciye verir. İstemci, bu bilgileri kullanarak doğrudan erişim token'ı almak için yetkilendirme sunucusuna başvurur. Bu tür genellikle, güvenilen uygulamalar için kullanılır, çünkü kullanıcı adı ve şifre istemciye doğrudan verilmiş olur. Bu nedenle güvenlik riskleri taşır ve sadece istemcinin tamamen güvenli olduğuna emin olunduğunda kullanılmalıdır.
- **Client Credentials Grant (İstemci Kimlik Doğrulama Türü)**: Bu tür, istemcinin kendisi adına bir erişim token'ı almasını sağlar. İstemci, kendi kimliğini doğrulamak için client id ve client secret bilgilerini kullanır. Bu tür genellikle istemci uygulamalarının API'lere erişmek için kullanılır, çünkü kullanıcı bilgisi gerekmez. Örneğin, bir mikroservis API'si, sadece istemci uygulamanın kimliğini doğrulayarak kendi kaynaklarına erişebilir.

3. Extension Grant Types (Uzantı Yetkilendirme Türleri)

OAuth 2.0, ek yetkilendirme türlerinin tanımlanmasına olanak tanır. Yani, OAuth 2.0'un sunduğu dört ana grant türüne ek olarak, yeni türler tanımlanabilir. Bu türler, OAuth 2.0 protokolünün genişletilebilirliğini sağlar ve özel ihtiyaçları karşılayacak şekilde yapılandırılabilir.

4. Authorization Grant Süreci

OAuth 2.0'daki yetkilendirme süreci temelde şu şekilde işler:

- İstemci, **authorization grant** almak için bir yetkilendirme türü seçer ve kullanıcıyı yetkilendirme sunucusuna yönlendirir.
- Kullanıcı, yetkilendirme sunucusunda kimlik doğrulaması yapar ve istemcinin isteklerine onay verir.
- Yetkilendirme sunucusu, istemciye bir **authorization grant** (yetki izni) sağlar.
- İstemci, **authorization grant**'i kullanarak erişim token'ı almak için **token endpoint**'ine başvurur.

Bu süreç, istemcilerin erişim token'ları alabilmesi için güvenli bir şekilde yetkilendirilmesini sağlar ve OAuth 2.0 protokolü, istemcilerin sadece gerekli kaynaklara erişmesini garanti eder.

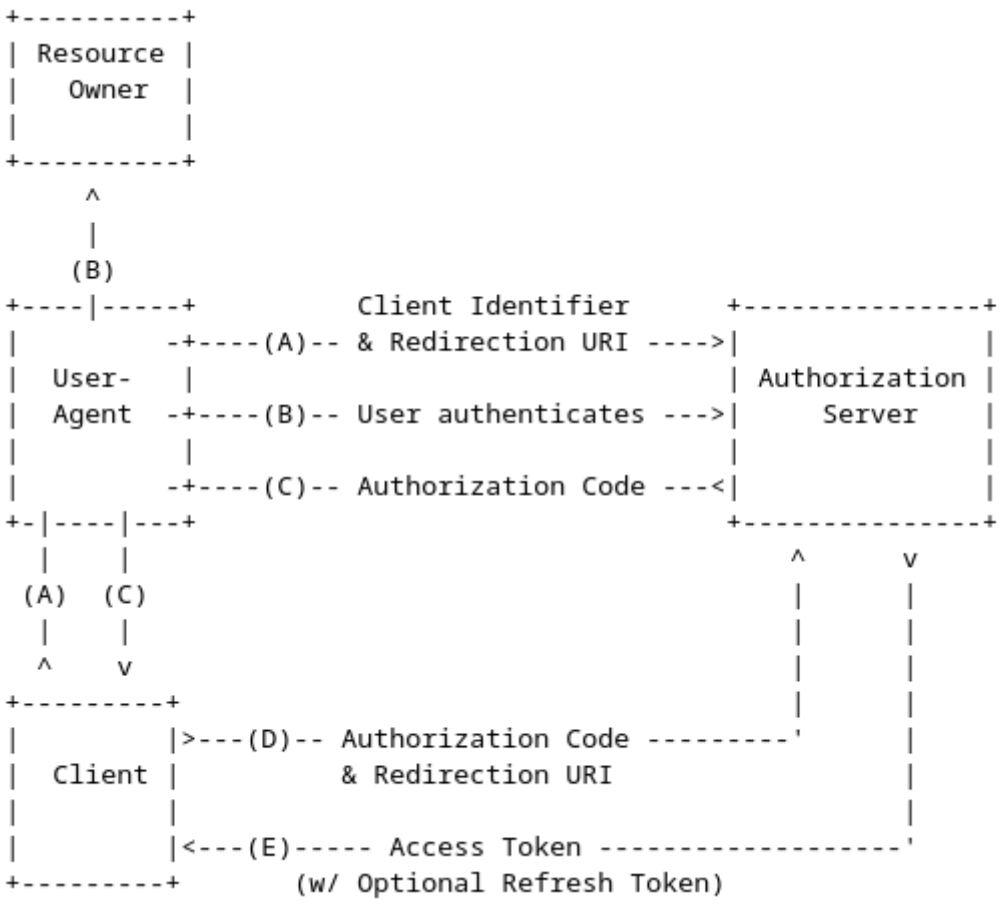
Özetle:

- OAuth 2.0'da **authorization** almak için istemci, kullanıcının onayını alarak **authorization grant** elde eder.
- Bu grant, istemcinin erişim token'ı almak için kullanacağı bir türdür.
- OAuth, istemcinin farklı kullanım senaryolarına göre dört ana grant türü ve uzantı türleri sunar.

Bu yapı, OAuth 2.0'ı esnek, güvenli ve farklı senaryolara uyarlanabilir bir yetkilendirme protokolü haline getirir.

4.1. Authorization Code Grant (Yetkilendirme Kodu Türü) bölümü, OAuth 2.0'da istemcilerin erişim token'ı ve tazeleme (refresh) token'ı almak için kullandığı bir yetkilendirme türünü açıklar. Bu tür, özellikle **confidential clients** (gizli istemciler) için optimize edilmiştir ve bir **redirection-based flow** (yönlendirme tabanlı akış) kullanır. Yani, istemci, kullanıcının bir web tarayıcısı aracılığıyla yetkilendirme sunucusuna yönlendirilmesini sağlar ve ardından kullanıcı onayı alındığında bir yetkilendirme kodu (authorization code) alır. Bu kod daha sonra erişim token'ı almak için kullanılır.

Bu akışın her aşamasını aşağıdaki gibi adım adım inceleyelim:



1. (A) Akışın Başlatılması

İstemci, **resource owner's user-agent** (kaynak sahibinin kullanıcı ajanı, genellikle bir web tarayıcısı) aracılığıyla yetkilendirme sunucusunun yetkilendirme endpoint'ine bir istek gönderir. Bu istekle birlikte aşağıdaki bilgiler iletilir:

- **Client Identifier**: İstemcinin kimliğini tanımlayan benzersiz bir etiket (client_id).
- **Requested Scope**: İstemcinin erişmek istediği kaynakların kapsamı.
- **Local State**: İstemcinin, kullanıcı ile etkileşim sırasında güvenliği sağlamak için kullanabileceği durum bilgisi (örneğin, CSRF saldırılarına karşı).
- **Redirection URI**: Kullanıcı yetkilendirmeyi tamamladıktan sonra, yetkilendirme sunucusunun kullanıcının tarayıcısını yönlendireceği URI. Bu URI, istemcinin kaydettiği ve doğrulamak için kullandığı URI ile eşleşmelidir.

2. (B) Kullanıcı Kimlik Doğrulaması ve Onay

Yetkilendirme sunucusu, **resource owner** (kaynak sahibi, yani kullanıcı) kimliğini doğrular ve istemcinin talep ettiği erişim izinlerine dair kullanıcıdan onay alır. Bu adımda, kullanıcıya genellikle erişim taleplerinin ne olduğunu ve bu talepleri kabul edip etmeyeceklerini belirten bir ekran gösterilir. Kullanıcı, talepleri onaylarsa, akış devam eder; reddederse, işlem sonlanır.

3. (C) Yetkilendirme Kodu İle Yönlendirme

Eğer kaynak sahibi erişim talebini onaylarsa, yetkilendirme sunucusu, kullanıcının tarayıcısını istemcinin verdiği **redirection URI**'ye yönlendirir. Yönlendirme sırasında:

- **Authorization Code** (Yetkilendirme Kodu): Yetkilendirme sunucusu tarafından verilen ve istemcinin daha sonra erişim token'ı almak için kullanacağı bir kod gönderilir.
- **Local State**: İstemcinin başlangıçta gönderdiği yerel durum bilgisi, ek güvenlik için redirection URI'siyle birlikte gönderilebilir.

4. (D) Erişim Token'ı Talebi

İstemci, aldığı **authorization code**'u kullanarak, **token endpoint**'e bir POST isteği gönderir. Bu istekle birlikte aşağıdaki bilgiler sunulur:

- **Authorization Code**: Kullanıcının onayıyla verilen yetkilendirme kodu.
- **Redirection URI**: Yetkilendirme kodunun alınmasında kullanılan redirection URI'si. Bu URI, doğrulama amacıyla tekrar gönderilir.

Bu adımda, istemci yetkilendirme sunucusuna **client authentication** (istemci kimlik doğrulaması) yapar. Yani istemci, kendini yetkilendirme sunucusuna tanıtarak doğrular (örneğin, client secret kullanılarak).

5. (E) Erişim Token'ı ve (Opsiyonel) Refresh Token'ın Alınması

Yetkilendirme sunucusu, istemcinin kimlik doğrulamasını yapar, **authorization code**'u doğrular ve **redirection URI**'yi kontrol eder. Eğer tüm bilgiler geçerli ise:

- **Access Token** (Erişim Token'ı): İstemciye erişim izni verilen kaynaklara ulaşması için gerekli olan token sağlanır.
- **Refresh Token** (Tazeleme Token'ı): Erişim token'ının süresi dolduğunda, istemcinin yenisini almak için kullanabileceği bir token sağlanabilir.

Bu adımlar tamamlandığında istemci, kullanıcı adına kaynaklara erişim için gerekli olan token'ları almış olur.

Akış Özeti

1. **İstemci**, kullanıcıyı yetkilendirme sunucusuna yönlendirir (authorization code flow başlatılır).
2. **Kullanıcı**, erişim talebini onaylar veya reddeder.
3. **Yetkilendirme Sunucusu**, kullanıcıyı istemcinin belirttiği URI'ye yönlendirirken bir **authorization code** sağlar.
4. **İstemci**, bu yetkilendirme kodunu kullanarak **access token** alır.

Bu Akışın Avantajları

- **Confidential Clients** için güvenlidir. Çünkü istemci, erişim token'ını almak için authorization code'u kullanır, bu da daha güvenli bir yol sunar.
- **Refresh token** ile erişim token'ının süresi dolduğunda kullanıcı tekrar kimlik doğrulama yapmak zorunda kalmaz.
- **Token güvenliği** açısından da faydalıdır, çünkü token doğrudan istemciye verilmez; önce bir authorization code kullanılır ve ardından erişim token'ı alınır.

Özetle

Authorization Code Grant, OAuth 2.0 protokolünde en güvenli ve yaygın kullanılan grant türüdür ve genellikle web uygulamaları gibi **confidential clients** tarafından tercih edilir. Bu akış, istemcinin daha sonra erişim token'ı alabilmesi için kullanıcıdan bir authorization code elde etmesine dayanır.

4.1.1. Authorization Request (Yetkilendirme İsteği) bölümü, istemcinin yetkilendirme sunucusuna yaptığı istekleri ve bu isteklerin doğru bir şekilde nasıl yapılandırılması gerektiğini açıklar. Bu bölümde, istemcinin yetkilendirme sunucusuna yapacağı HTTP isteği, gerekli parametreler ve isteklerin nasıl yapılması gerektiği belirtilmiştir.

Adım Adım Açıklama

1. **Authorization Request (Yetkilendirme İsteği) Oluşturma** İstemci, **authorization endpoint URI**'ye (yetkilendirme sunucusunun belirtilen URL'si) aşağıdaki parametrelerle bir URI (Uniform Resource Identifier) oluşturur. Bu URI'yi, istemci kullanıcının tarayıcısına yönlendirmek için kullanır. İstek, **application/x-www-form-urlencoded** formatında yapılır (bu, parametrelerin URL'nin sorgu kısmına eklenmesi anlamına gelir).

İstek Parametreleri

- **response_type** (Zorunlu): Değerinin **"code"** olması gerekir. Bu, istemcinin **authorization code** almak istediğini belirtir. Yani istemci, erişim token'ı almak için yetkilendirme kodu almak üzere bu parametreyi kullanır.
- **client_id** (Zorunlu): Bu, istemcinin kimliğini belirten benzersiz bir tanımlayıcıdır. İstemcinin yetkilendirme sunucusuna kaydedilmiş ve tanımlanmış olmalıdır. Bu parametre, istemcinin kimliğini yetkilendirme sunucusuna iletmek için kullanılır.
- **redirect_uri** (Opsiyonel): Bu, kullanıcı yetkilendirme işlemini tamamladıktan sonra, yetkilendirme sunucusunun kullanıcıyı geri yönlendireceği URI'dir. Eğer istemci daha önce kaydettiyse, bu URI'yi içermelidir. Eğer bu parametre sağlanmazsa, yetkilendirme sunucusu varsayılan URI'yi kullanabilir. Bu parametre, kullanıcı yetkilendirme işlemini tamamladığında, istemcinin doğru yere yönlendirilmesini sağlar.
- **scope** (Opsiyonel): İstemcinin erişim talep ettiği kaynakların kapsamını belirler. Örneğin, istemci sadece kullanıcı profilini mi almak istiyor, yoksa daha geniş bir erişim mi talep ediyor? Sunucu, istemcinin talep ettiği kapsamı dikkate alarak, erişim token'ı verirken belirli yetkilerle sınırlandırabilir.

- **state** (Tavsiye Edilir): Bu, istemci tarafından kullanılan, istek ve callback arasındaki durumu korumaya yarayan opak (şeffaf olmayan) bir değerdir. Yani, istemci bir istek gönderdiğinde, bu parametre istemcinin işlem durumu ile ilgili bilgileri tutar. Sunucu, kullanıcıyı yetkilendirme işlemi tamamlandıktan sonra yönlendirdiğinde, istemci bu **state** parametresini geri alır ve işlemde herhangi bir güvenlik açığı olmadan süreci yönetebilir. Bu parametre, **Cross-Site Request Forgery (CSRF)** saldırılarına karşı bir önlem olarak da kullanılır.

İstek Örneği

Örneğin, istemci kullanıcıyı şu şekilde bir HTTP GET isteği ile yönlendirebilir:

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

Bu istek, istemcinin yetkilendirme kodunu almak için yetkilendirme sunucusuna yaptığı ilk adımdır. Burada:

- `response_type=code`: Yetkilendirme kodunun istenmesini belirtir.
- `client_id=s6BhdRkqt3`: İstemcinin kimliğini belirtir.
- `state=xyz`: İstemcinin durumu (örneğin, güvenlik amaçlı bir koruma).
- `redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb`: Yetkilendirme işleminden sonra kullanıcının yönlendirileceği URI.

Sunucu Tarafında Yapılacak Doğrulamalar

Yetkilendirme sunucusu, istemciden gelen bu isteği aldıktan sonra şu doğrulamaları yapar:

- **Gerekli Parametrelerin Varlığı**: İstekten gerekli parametrelerin (response_type, client_id) eksik olmaması gerekir. Eğer bu parametrelerden biri eksikse, istek geçersiz sayılabilir.
- **Parametrelerin Geçerliliği**: Gelen parametrelerin geçerli olup olmadığı doğrulanır. Örneğin, client_id'nin doğru istemciyi temsil ettiğinden emin olunmalıdır.
- **Redirection URI'nin Kaydı**: Eğer bir **redirect_uri** sağlanmışsa, yetkilendirme sunucusu bu URI'yi istemcinin kaydına göre doğrular. URI'nin kayıtsız olması veya uyumsuzluk olması durumunda, istek geçersiz kabul edilebilir.

Sunucu İşlemleri

- **Kimlik Doğrulama ve Onay**: Yetkilendirme sunucusu, kaynak sahibini (kullanıcıyı) kimlik doğrulama işlemi yaparak onay alır. Bu işlem, kullanıcıdan parolalarını girmelerini veya başka kimlik doğrulama yöntemlerini kullanmalarını içerebilir.
- **Yönlendirme**: Eğer kullanıcı onay verirse, yetkilendirme sunucusu, kullanıcıyı belirlenen **redirect_uri**'ye yönlendirir. Bu yönlendirme ile birlikte, kullanıcının onayına dair veriler ve **authorization code** (yetkilendirme kodu) gönderilir.

Sonuç

Authorization Request bölümü, istemcinin yetkilendirme sunucusuna, kullanıcı adına erişim talep etmek için yaptığı ilk adımdır. Bu adımda, istemci gerekli parametrelerle bir URI oluşturur ve

kullanıcının tarayıcısını bu URI'ye yönlendirir. Sunucu, parametreleri doğruladıktan sonra, kullanıcının onayını alır ve istemciyi doğru URI'ye yönlendirir.

4.1.2. Authorization Response *(Yetkilendirme Yanıtı) bölümü, yetkilendirme sunucusunun, istemcinin yetkilendirme isteğini kabul etmesi durumunda yaptığı yanıtı ve yanıtın nasıl yapılandırılacağını açıklar. Bu bölüm, istemcinin aldığı yetkilendirme kodunu içeren yanıtın formatını ve gerekli parametreleri tanımlar.*

Adım Adım Açıklama

1. **Authorization Response (Yetkilendirme Yanıtı) Oluşumu**

   Yetkilendirme sunucusu, **yetkilendirme isteğini onayladıktan sonra** istemciye bir **yetkilendirme kodu** (authorization code) gönderir. Bu, istemcinin daha sonra erişim token'ı alabilmesi için kullanacağı bir koddur. Yanıt, istemcinin belirlediği **redirection URI**'ye yapılacak bir yönlendirme ile gerçekleşir. Yönlendirme URL'si aşağıdaki parametreleri içerir:

Yanıt Parametreleri

- **code** (Zorunlu): Bu, yetkilendirme sunucusunun oluşturduğu **yetkilendirme kodu**'dur. İstemci bu kodu, ilerleyen adımlarda erişim token'ı almak için kullanacaktır.
    - **Geçerlilik Süresi**: Yetkilendirme kodu, kullanıldıktan sonra çok kısa bir süre içinde (maksimum 10 dakika) geçerliliğini yitirmelidir. Bu, güvenlik amacıyla, kodun sızması durumunda kötüye kullanılmasını engellemek için önemlidir.
    - **Tek Kullanımlık**: Yetkilendirme kodu yalnızca bir kez kullanılabilir. Eğer bir yetkilendirme kodu birden fazla kez kullanılırsa, yetkilendirme sunucusu isteği reddeder ve mümkünse, bu koda dayalı olarak verilen tüm token'ları iptal eder.
    - **Bağlı Olma**: Yetkilendirme kodu, istemcinin kimliğine (client_id) ve kullanılan **redirection URI**'ye bağlıdır. Yani, yalnızca doğru istemci ve doğru URI ile kullanılabilir.
- **state** (Zorunlu, Eğer "state" Parametresi İstemci İsteğinde Varsa): Eğer istemci, yetkilendirme isteği sırasında bir **state** parametresi gönderdiysa, yetkilendirme sunucusu aynı değeri yanıt olarak göndermelidir. Bu parametre, istemcinin gönderdiği **state** değerini geri almasını sağlar. **State** parametresi, özellikle **CSRF (Cross-Site Request Forgery)** saldırılarını engellemek amacıyla kullanılır.

Yönlendirme Yanıtı Örneği

Yetkilendirme sunucusunun istemciye verdiği yanıt, aşağıdaki gibi bir HTTP yönlendirme yanıtı olabilir:

HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA
&state=xyz

Burada:

- **Location** başlığı, istemcinin yönlendirilmesi gereken URI'yi belirtir.
- **code=SplxlOBeZQQYbYS6WxSbIA**: Bu, yetkilendirme sunucusunun verdiği yetkilendirme kodudur. İstemci, bu kodu alarak erişim token'ı almak için kullanacaktır.
- **state=xyz**: Eğer istemci, isteğinde **state** parametresi göndermişse, bu değer burada geri gönderilir.

Sunucu Tarafında Yapılacak Doğrulamalar

- **Yetkilendirme Kodu**: Sunucu, istemcinin geçerli bir yetkilendirme kodu alıp almadığını kontrol eder. Bu kod, yalnızca bir kez kullanılabilir ve istemcinin doğru kimlik bilgileri ve URI ile istek yapmış olması gerekir.
- **State Parametresi**: Eğer istemci bir **state** parametresi göndermişse, yetkilendirme sunucusu bu parametreyi geri gönderecektir. Bu, istemcinin doğru bir işlem yaptığını doğrulaması için gereklidir.
- **Geçerlilik Süresi**: Yetkilendirme kodunun geçerlilik süresi sınırlıdır. Genellikle 10 dakika içinde geçerliliğini yitirir. Bu, saldırganların kodu ele geçirmesini ve kötüye kullanmasını engellemek için önemlidir.

İstemci Tarafında Yapılacaklar

- **Yanıt Parametrelerini İşleme**: İstemci, yetkilendirme sunucusundan gelen **code** (yetkilendirme kodu) ve **state** parametrelerini işlemelidir. Eğer sunucu tanımadığı parametreleri gönderirse, istemci bu parametreleri göz ardı etmelidir.
- **Authorization Code Kullanımı**: İstemci, aldığı **yetkilendirme kodunu** doğrulayıp, bir **access token** almak için token endpoint'e bir istek yapar. Bu noktada istemci, doğru istemci kimliği ve doğru **redirection URI** ile kodu kullanmalıdır.

Sonuç

Authorization Response bölümü, yetkilendirme sunucusunun istemciye **yetkilendirme kodu** verdiği yanıt sürecini açıklar. Bu süreç, istemcinin doğru bir şekilde yetkilendirilip edilmediğini ve yönlendirme URI'sine ne tür parametrelerin eklenmesi gerektiğini belirler. Sunucu, geçerli bir kod ve istemci tarafından sağlanan **state** parametresini geri göndererek, istemcinin güvenli bir şekilde sonraki adımlara geçmesini sağlar.

***4.1.2.1. Error Response** (Hata Yanıtı) bölümü, istemcinin yetkilendirme isteği sırasında bir hata meydana gelmesi durumunda, yetkilendirme sunucusunun nasıl bir yanıt vereceğini açıklar. Bu yanıt, istemcinin hata durumunu doğru şekilde anlaması ve uygun şekilde işlem yapabilmesi için gerekli bilgileri içerir.*

Hata Durumları ve Yanıtları

Hata yanıtı, istemcinin yetkilendirme isteği sırasında karşılaştığı sorunları belirtmek için kullanılır. Eğer bir hata oluşursa, **yetkilendirme sunucusu**, istemciyi doğrudan geçersiz **redirection URI**'ye yönlendirmez. Bunun yerine, hata mesajlarını içeren bir HTTP yanıtı gönderir. Hata yanıtı,

istemcinin daha sonra hatanın nedenini anlaması ve gerekli düzeltmeleri yapabilmesi için gereken bilgileri içerir.

Aşağıda, hata durumları ve her bir hata türü için kullanılan parametreler açıklanmıştır.

Hata Parametreleri

1. **error** (Zorunlu): Hata kodu, isteğin neden başarısız olduğunu belirtir. Bu kodlar, hata türüne göre belirlenir. Bu hata kodları, yalnızca belirli ASCII karakterlerden oluşmalıdır ve aşağıdaki değerlerden biri olabilir:
    - **invalid_request**: İstek eksik bir parametre içeriyor, geçersiz bir parametre değeri var, bir parametre birden fazla kez kullanılmış veya istek biçimsel olarak yanlış.
    - **unauthorized_client**: İstemci, belirtilen yetkilendirme kodunu almak için yetkilendirilmemiştir.
    - **access_denied**: Kaynak sahibi ya da yetkilendirme sunucusu, erişim isteğini reddetmiştir.
    - **unsupported_response_type**: Yetkilendirme sunucusu, belirtilen yöntemle yetkilendirme kodu almak için destek sunmuyor.
    - **invalid_scope**: İstenilen kapsam (scope), geçersiz, bilinmeyen veya hatalı biçimlendirilmiş.
    - **server_error**: Yetkilendirme sunucusu, isteği yerine getiremeyen beklenmeyen bir durumla karşılaştı.
    - **temporarily_unavailable**: Yetkilendirme sunucusu geçici bir yoğunluk veya bakım nedeniyle isteği işleyemiyor.

    Not: `error` parametresinin değeri yalnızca belirli karakterlerden oluşmalıdır ve bu karakterler şu setle sınırlıdır: `%x20-21 / %x23-5B / %x5D-7E`.
2. **error_description** (Opsiyonel): İnsan tarafından okunabilir, ASCII formatında açıklamalar sağlayan bir metin. Bu açıklama, geliştiricinin hatayı anlamasında yardımcı olur. **error_description** parametresi de yalnızca belirli ASCII karakterleri içerebilir.
3. **error_uri** (Opsiyonel): Hata hakkında daha fazla bilgi sağlamak için bir URI (Uniform Resource Identifier) içerir. Bu URI, hata hakkında daha fazla bilgi almak için kullanılabilir. Bu parametre, **error_description** ile birlikte verilebilir. **error_uri** parametresi de belirli karakterlerden oluşmalıdır ve URI referansı biçimine uygun olmalıdır.
4. **state** (Zorunlu, eğer istemci isteği sırasında **state** parametresi kullanılmışsa): Eğer istemci, yetkilendirme isteği sırasında bir **state** parametresi göndermişse, yetkilendirme sunucusu bu değeri **state** parametresi olarak geri göndermelidir. Bu, istemcinin hangi isteğe karşılık gelen hatayı aldığını doğru şekilde eşleştirmesine yardımcı olur. **state** parametresi, istemcinin göndermiş olduğu orijinal değeri içerir.

Hata Yanıtı Örneği

Eğer bir hata meydana gelirse, yetkilendirme sunucusu aşağıdaki gibi bir yönlendirme yanıtı verebilir:

HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied&state=xyz

Burada:

- **error=access_denied**: Bu, kaynağa erişim isteğinin reddedildiğini belirten hata kodudur.
- **state=xyz**: Eğer istemci, isteğinde bir **state** parametresi göndermişse, yetkilendirme sunucusu bu değeri geri gönderecektir.

Hata Durumlarının Kullanımı

Hata Parametreleri istemcinin doğru bir şekilde hatayı anlamasını sağlar ve hatayı nasıl ele alacağını belirlemesine yardımcı olur. Örneğin:

- Eğer **invalid_request** hatası alınırsa, istemci istek parametrelerini gözden geçirmelidir (eksik parametre, yanlış değer, çoklu parametre vb.).
- Eğer **access_denied** hatası alınırsa, kaynak sahibi ya da sunucu, isteği reddetmiştir. Bu durumda istemci, kullanıcıdan yeniden izin almayı deneyebilir.
- **server_error** ve **temporarily_unavailable** gibi hata durumları genellikle sunucu tarafındaki sorunlardan kaynaklanır ve istemciye bu durumda tekrar denemesi gerektiği bildirilebilir.

Sonuç

Error Response bölümü, yetkilendirme isteğinde hata olması durumunda istemciye nasıl bir yanıt verileceğini tanımlar. İstemciye hata kodu ve açıklaması, sorunları çözebilmesi için gerekli bilgileri sağlar. Bu, OAuth akışında güvenliği ve doğru işlem sırasını sağlamak için kritik bir adımdır.

***4.1.3. Access Token Request** (Erişim Token'ı İsteği) bölümü, istemcinin yetkilendirme kodunu alarak bir erişim token'ı almak için token endpoint'ine nasıl başvuracağını açıklar. Erişim token'ı, istemcinin kullanıcı adına bir kaynağa erişmesine izin verir ve genellikle bu işlem **authorization code flow** içinde yapılır. Aşağıda, bu isteğin nasıl yapılacağı ve hangi parametrelerin gerekli olduğu detaylı olarak açıklanmıştır.*

Erişim Token'ı İsteği için Gerekli Parametreler

Erişim token'ı almak için istemcinin, yetkilendirme kodu ile bir istek yapması gerekmektedir. Bu istek **"application/x-www-form-urlencoded"** formatında yapılır ve aşağıdaki parametreler içerir:

1. **grant_type** (Zorunlu): İstemci, erişim token'ı almak için hangi türde yetkilendirme istediğini belirtir. Bu parametre mutlaka **"authorization_code"** olarak ayarlanmalıdır, çünkü bu akışta istemci bir yetkilendirme kodu alarak erişim token'ı talep etmektedir.
2. **code** (Zorunlu): İstemcinin, yetkilendirme sunucusundan aldığı ve daha önce **authorization code flow** sırasında elde ettiği yetkilendirme kodunu belirtir. Bu kod, istemcinin token almak için yapacağı istekte kullanılır.
3. **redirect_uri** (Zorunlu): Eğer istemci, yetkilendirme isteği sırasında bir **redirect_uri** parametresi belirtmişse, bu parametre de bu istekte yer almalıdır. Bu değerin, ilk yetkilendirme isteğinde kullanılan **redirect_uri** ile tam olarak eşleşmesi gerekir. Bu, güvenlik için yapılan bir kontrol olup, istemcinin doğru bir URI'yi kullanmasını sağlar.

4. **client_id** (Zorunlu): Eğer istemci, **client authentication** yapmadan yetkilendirme kodunu almışsa, bu parametre de gerekli olabilir. Bu, istemcinin kimliğini doğrulamak için kullanılır.
5. **client authentication** (Zorunlu, Eğer Gizli İstemci): Eğer istemci gizli (confidential) bir istemci ise veya istemci kimlik doğrulaması gerektiriyorsa, istemci bu isteği gönderirken kimlik doğrulama yapmalıdır. Bu genellikle istemcinin bir API anahtarı veya başka bir kimlik doğrulama yöntemi ile yapılır. İstemci, kimlik doğrulaması için **Basic Authentication** gibi yöntemler kullanabilir. Kimlik doğrulaması başvurusu, istemcinin kimliğini doğrulamak için kullanılır.

   Örnek olarak, istemcinin yaptığı bir istek şu şekilde görünebilir:

   POST /token HTTP/1.1
   Host: server.example.com
   Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
   Content-Type: application/x-www-form-urlencoded

   grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
   &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

Yetkilendirme Sunucusunun Yapması Gerekenler

Yetkilendirme sunucusu, istemcinin başvurusunu işleme alırken aşağıdaki doğrulama ve kontrol işlemlerini yapmalıdır:

1. **Client Authentication Gereksinimi**: Eğer istemci gizli bir istemci (confidential client) ise ya da istemci kimlik doğrulama bilgilerine sahipse, yetkilendirme sunucusu istemcinin kimliğini doğrulamalıdır. Bu doğrulama, genellikle istemcinin kimlik doğrulama başlığı (örneğin **Basic Authentication**) kullanarak yapılır.
2. **Authorization Code Kontrolü**: Yetkilendirme sunucusu, başvuruyu yapan istemciye verilen yetkilendirme kodunun doğruluğunu kontrol etmelidir. Bu, verilen kodun geçerli olduğunu ve sunucu tarafından doğru bir şekilde oluşturulduğunu doğrular.
3. **Redirect URI Doğrulaması**: Eğer istemci, yetkilendirme isteği sırasında bir **redirect_uri** belirtmişse, yetkilendirme sunucusu bu URI'nin doğruluğunu ve geçerliliğini kontrol etmelidir. **redirect_uri** parametresi, ilk yetkilendirme isteğinde kullanılan **redirect_uri** ile tam olarak eşleşmelidir.
4. **Client ID Doğrulaması**: Yetkilendirme sunucusu, istemcinin kimliğini doğrulamalıdır. Eğer istemci gizli bir istemci ise ve kimlik doğrulaması yapılmışsa, yetkilendirme sunucusu istemcinin doğru kimliğe sahip olduğundan emin olmalıdır. Eğer istemci halka açık (public) bir istemci ise, sunucu, yetkilendirme kodunun doğru istemciye verildiğini doğrulamalıdır.

Sonuç

Bu işlem, istemcinin **yetkilendirme kodu** alarak **erişim token'ı** almasını sağlayan önemli bir adımdır. Yetkilendirme sunucusu, istemciden gelen talepleri doğrulamalı ve güvenli bir şekilde erişim token'larını sağlamalıdır. İstemcinin yaptığı istek, doğru parametrelerle yapılmalı ve güvenlik önlemleri (örneğin, **redirect_uri** doğrulaması, **client authentication** vb.) uygulanmalıdır. Bu sayede, istemci yalnızca yetkilendirilen ve geçerli bir token alır.

***4.1.4. Access Token Response** (Erişim Token'ı Yanıtı) bölümü, istemcinin **access token** almak için yaptığı başvurunun başarılı bir şekilde işlendiğinde yetkilendirme sunucusunun nasıl yanıt vereceğini açıklar. Bu yanıt, istemcinin güvenli bir şekilde kaynaklara erişebilmesi için gerekli olan **access token**'ı içerir. Eğer istemci kimlik doğrulaması hatalıysa veya geçersizse, bir hata yanıtı döner.*

Erişim Token'ı Yanıtı İçin Gerekli Parametreler

Başarılı bir erişim token'ı yanıtı şu parametreleri içerir:

1. **access_token** (Erişim Token'ı):
    - Zorunludur. Bu, istemcinin kimliğini doğrulayan ve kaynağa erişim sağlamak için kullanacağı **erişim token'ı**'dır.
    - Erişim token'ı, kaynak sunuculara yapılan isteklerde kullanılacak ve istemcinin kaynaklara erişmesine izin verecektir.
    - Bu token, genellikle bir **JWT** (JSON Web Token) ya da başka bir şifreli formatta olabilir.
2. **token_type** (Token Türü):
    - Zorunludur. Erişim token'ının türünü belirtir. Genellikle **"Bearer"** kullanılır, bu da istemcinin, token'ı "Bearer" başlığı altında HTTP isteklerinde göndermesi gerektiğini ifade eder.
    - Örneğin, **"Bearer"** token türü kullanıldığında, istemci HTTP isteklerinde şu şekilde bir başlık gönderir:

      ```
      Authorization: Bearer <access_token>
      ```

3. **expires_in** (Süresi Dolacak Zaman):
    - Zorunludur. Erişim token'ının geçerliliği için kalan süreyi saniye cinsinden belirtir. Genellikle **3600 saniye** (1 saat) gibi bir değer verilir.
    - Token süresi dolduğunda, istemci **refresh token** kullanarak yeni bir erişim token'ı alabilir (eğer **refresh_token** verilmişse).
4. **refresh_token** (Opsiyonel, Yenileme Token'ı):
    - Opsiyoneldir. Bu token, istemcinin erişim token'ı süresi dolduğunda yeni bir erişim token'ı alabilmesi için kullanılır.
    - **Refresh token** genellikle uzun süre geçerli olur ve kullanıcının yeniden giriş yapmasını engellemek amacıyla token yenileme işlemi için kullanılır. Eğer istemciye **refresh_token** verilirse, bu token başka bir istekte kullanılarak erişim token'ı yenilenebilir.
5. **example_parameter** (Opsiyonel, Örnek Parametre):
    - Opsiyoneldir. Yetkilendirme sunucusu, başka ek parametreler ekleyebilir. Bu parametre, örneğin belirli bir özellik ya da uygulamaya özel bir bilgi olabilir.

Örnek Başarılı Yanıt

Başarılı bir erişim token'ı yanıtı aşağıdaki gibi görünebilir:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter": "example_value"
}
```

Yanıtın Açıklamaları:

- **access_token**: Bu, istemcinin kullanacağı ve kaynaklara erişim sağlayacak token'dır. Örneğin, **2YotnFZFEjr1zCsicMWpAA**.
- **token_type**: Bu örnekte **"example"** olarak belirtilmiş. Gerçek dünyada, çoğu zaman **"Bearer"** olacaktır.
- **expires_in**: Bu token'ın 3600 saniye, yani 1 saat geçerli olduğu belirtilmiş.
- **refresh_token**: Eğer istemci, erişim token'ı süresi dolduğunda yeni bir token almak isterse, bu **refresh_token** kullanılabilir. Bu örnekte **tGzv3JOkF0XG5Qx2TlKWIA**.
- **example_parameter**: Bu parametre, yetkilendirme sunucusu tarafından eklenen bir ek parametre olabilir. Bu tür parametreler, belirli özelliklere ya da kullanıcılara özgü olabilir.

Hata Durumu

Eğer istemcinin isteği geçersizse veya kimlik doğrulaması başarısız olursa, yetkilendirme sunucusu bir **hata yanıtı** dönecektir. Bu hata yanıtında, hata tipi ve nedenini belirten **error** parametresi bulunur. Örneğin, geçersiz bir **client_id** veya hatalı bir **authorization code** verildiğinde hata döner.

Sonuç olarak, başarılı bir **Access Token Response** yanıtı, istemcinin gerekli erişim yetkilerine sahip olabilmesi için kritik öneme sahiptir. Erişim token'ı, istemcinin kaynağa erişmesine olanak tanırken, yenileme token'ı da uzun süreli oturumlar için gereklidir. Bu yanıt, istemci ile sunucu arasındaki güvenli ve sürekli veri erişiminin sağlanması için kullanılır.

4.2. Implicit Grant (Açık İzin Yöntemi) bölümü, **Access Token** almak için kullanılan bir OAuth 2.0 yetkilendirme akışını tanımlar. **Implicit Grant**, genellikle istemcilerin (genellikle JavaScript gibi istemci tarafı dillerini kullanan) tarayıcıda çalışan ve belirli bir redirection URI'ye sahip olan halka açık istemciler (public clients) için optimize edilmiştir. Bu akış, istemci kimlik doğrulaması gerektirmeyen bir yapıya sahiptir ve erişim token'ı doğrudan yetkilendirme isteğinin sonucu olarak elde edilir.

Bu yöntem, **Authorization Code Grant** akışından farklıdır. **Authorization Code Grant** akışında istemci, önce yetkilendirme kodunu alır, ardından bunu kullanarak bir erişim token'ı talep eder. Ancak **Implicit Grant**'te, istemci doğrudan yetkilendirme isteğiyle erişim token'ını alır.

Akışın Genel Özeti

1. **Client (İstemci)**, yetkilendirme sunucusuna bir erişim isteği gönderir. Bu isteği göndermek için istemci, yetkilendirme sunucusuna yönlendirdiği **resource owner** (kaynak sahibi) kullanıcısından **client identifier**, **requested scope**, **local state** ve bir **redirection URI** (geri yönlendirme URI) içerir. Bu URI, kaynak sahibi (kullanıcı) erişim iznini verdikten sonra kullanıcıyı geri yönlendirmek için kullanılır.
2. **Authorization Server (Yetkilendirme Sunucusu)**, kullanıcıyı kimlik doğrulamak için kullanıcı aracı (tarayıcı) üzerinden işlemi başlatır. Kullanıcı, istemcinin erişim isteğini onaylar veya reddeder.
3. Eğer kullanıcı erişimi onaylarsa, yetkilendirme sunucusu kullanıcıyı istemciye belirlenen redirection URI'sine geri yönlendirir. Bu URI, erişim token'ını **URI fragment** içinde içerir. **URI fragment**, URL'nin "?" ile başlayan sorgu parametrelerinden farklıdır. Bu parametreler genellikle URL'nin son kısmında yer alır ve kullanıcı aracısı tarafından işlenir.
4. Kullanıcı aracı, verilen redirection URI'yi takip eder ve **fragment**'i yerel olarak saklar. Ancak bu bilgi, kullanıcı aracı tarafından yalnızca tarayıcıda saklanır ve web sunucusuna gönderilmez.
5. Web barındırma istemcisi (örneğin, bir JavaScript uygulaması), verilen URI'den **fragment**'i çıkarır ve **access token**'ı (erişim token'ını) çıkarabilmek için ilgili scripti kullanarak bu veriyi alır.
6. Kullanıcı aracı, script tarafından çıkarılan **access token**'ı istemciye iletir.

Akışın Aşamaları

1. **(A) İstemci Başlangıç**:
    - İstemci, **resource owner** (kullanıcı) aracıyla yetkilendirme sunucusuna yönlendirme yapar. İstemci bu adımda **client identifier**, **requested scope** (istek kapsamı), **local state** (yerel durum) ve bir **redirection URI** gönderir.
2. **(B) Kimlik Doğrulama**:
    - Yetkilendirme sunucusu, kullanıcıyı kimlik doğrulamak için işlem yapar. Kullanıcı, istemcinin erişim talebini kabul eder veya reddeder.
3. **(C) Yetkilendirme Sunucusunun Yönlendirmesi**:

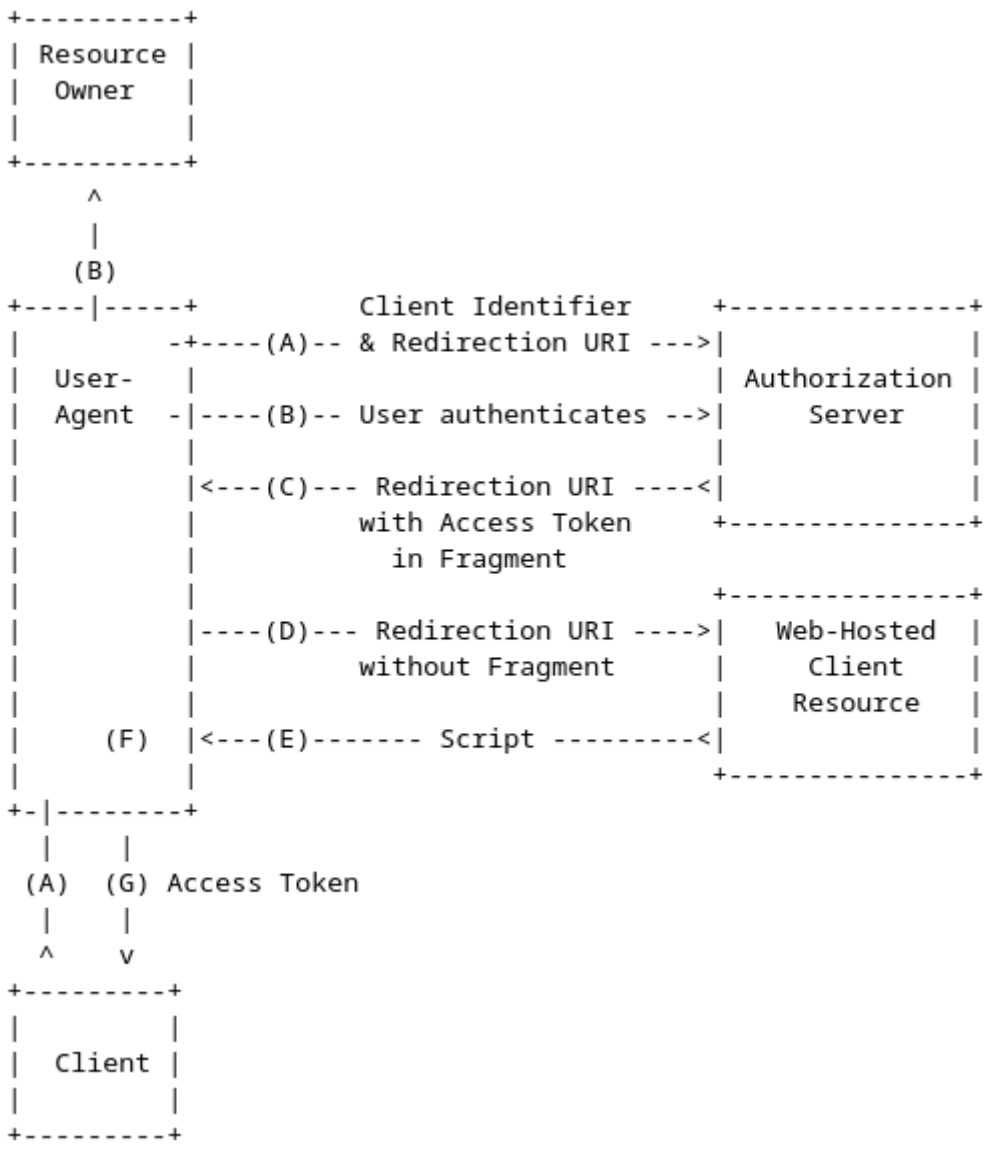
    - Kullanıcı erişimi onaylarsa, yetkilendirme sunucusu kullanıcıyı verilen **redirection URI**'ye yönlendirir. Bu URI, erişim token'ını **fragment** içinde içerir.
4. **(D) Yönlendirme ve Fragment Saklama**:
    - Kullanıcı aracı, **redirection URI**'yi takip eder ancak **fragment** bilgisini sadece yerel olarak saklar. Web sunucusu bu bilgiyle ilgili herhangi bir işlem yapmaz.
5. **(E) Web-Hosted Client Scripti**:
    - Web barındıran istemci, verilen **redirection URI**'yi işler. Web istemcisi, bu URI'deki **fragment**'i çıkarabilir ve gerekli parametreleri elde edebilir (örneğin, **access token**).
6. **(F) Scriptin Çalıştırılması**:
    - Kullanıcı aracı, scripti çalıştırarak **access token**'ı çıkarır.
7. **(G) Access Token'ın İstemciye Gönderilmesi**:
    - Kullanıcı aracı, çıkarılan **access token**'ı istemciye iletir.

Implicit Grant Yöntemiyle İlgili Önemli Noktalar

- **Refresh Token Yok**: Implicit Grant yöntemi, refresh token desteklemez. Erişim token'ı yalnızca tek bir kez kullanılır ve süresi dolduğunda yenilenmesi gerekebilir. Bu durumda, kullanıcı tekrar kimlik doğrulaması yaparak yeni bir token almalıdır.
- **Güvenlik Riskleri**: Erişim token'ı, URI fragmentinde açıkça yer alır ve bu da güvenlik riski oluşturabilir. Token, kullanıcının tarayıcısında ya da başka uygulamalarla paylaşılabilir. Bu nedenle, **Implicit Grant** yöntemi, güvenliği çok daha kritik olan işlemler için önerilmez. Bunun yerine, **Authorization Code Grant** yöntemi daha güvenli bir alternatif olabilir.
- **Public Clients için Uygun**: Bu akış genellikle, istemcinin kimlik doğrulaması yapmadığı ve istemcinin yalnızca kullanıcının tarayıcısında çalışan, güvenli olmayan ortamlar için uygundur.

Akışın Şeması

Aşağıda **Implicit Grant Flow**'unun adımları daha görsel bir şekilde gösterilmiştir:



Bu görseldeki adımlar sırasıyla:

(A) İstemcinin Başlatması

İstemci, yetkilendirme akışını başlatmak için **resource owner**'ı (kaynak sahibi kullanıcı) yetkilendirme sunucusunun **authorization endpoint**'ine (yetkilendirme noktası) yönlendirir. Bu aşamada, istemci aşağıdaki bilgileri içerir:

- **Client Identifier (İstemci Kimliği)**: İstemcinin kimliğini tanımlar. Bu, genellikle istemcinin kayıtlı olduğu benzersiz bir değerdir.
- **Requested Scope (İstenen Kapsam)**: İstemcinin erişmek istediği kaynakların veya izinlerin listesi. Örneğin, kullanıcı bilgilerine erişim gibi.
- **Local State (Yerel Durum)**: Bu, istemcinin talebine dair geçici bir durum bilgisidir ve CSRF (Cross-Site Request Forgery) saldırılarına karşı koruma sağlamak için kullanılır.

- **Redirection URI (Yönlendirme URI'si)**: Yetkilendirme sunucusunun, kullanıcıyı başarılı bir şekilde yetkilendirdikten sonra geri yönlendireceği URI. Bu, istemcinin doğru kaynağa yönlendirilmesini sağlamak için gereklidir.

Bu aşama, istemcinin kullanıcıyı yetkilendirme sunucusuna yönlendirdiği ilk adımdır. Yönlendirme URI'si, daha sonra kullanıcı başarılı bir şekilde giriş yaparsa erişim token'ı ile birlikte dönecek olan yerdir.

(B) Yetkilendirme Sunucusunun Kimlik Doğrulaması

Yetkilendirme sunucusu, kullanıcının kimliğini doğrulamak için **user-agent** (genellikle bir web tarayıcısı) üzerinden işlem yapar. Bu adımda:

- Kullanıcı, yetkilendirme sunucusuna girerek, istemcinin talep ettiği erişim izni ile ilgili bir karar verir.
- Kullanıcı, istemcinin erişim talebini **onaylar** veya **reddeder**. Bu aşama, kullanıcının istemcinin hangi verilere erişebileceği konusunda karar verdiği aşamadır.

Eğer kullanıcı **erişim iznini reddederse**, yetkilendirme sunucusu istemciyi hatalı bir yanıtla (örneğin, hata kodu) yönlendirir.

(C) Yetkilendirme Sunucusunun Yönlendirmesi

Eğer kullanıcı, istemcinin erişim talebini **onaylarsa**, yetkilendirme sunucusu, kullanıcıyı tekrar istemciye yönlendirecektir. Bu yönlendirme şu şekilde gerçekleşir:

- Yetkilendirme sunucusu, **redirection URI**'yi kullanarak kullanıcıyı geri yönlendirir.
- Bu URI, **access token**'ı **fragment** içinde içerir. Fragment kısmı, URL'nin sonundaki # ile başlar ve tarayıcıda görünür ama sunucuya gönderilmez.
-

Bu akış, istemcinin kimlik doğrulaması yapmadan, doğrudan tarayıcıda çalışan istemciler için geçerli olan ve hızlıca erişim sağlayan bir mekanizma sağlar.

Bu aşama, erişim token'ının doğrudan kullanıcı aracı (tarayıcı) aracılığıyla istemciye iletilmesinin sağlandığı adımdır.

(D) Kullanıcı Aracısının Yönlendirmeyi Takip Etmesi

Kullanıcı, yönlendirme işlemini takip ederek istemciye geri gelir. Ancak:

- **Fragment** bilgisi (yani access token), URL'nin bir parçası olarak kullanıcı aracısında (tarayıcıda) saklanır ve **web sunucusuna gönderilmez**.
- Yönlendirme, istemcinin web kaynağına yapılır (örneğin, HTML sayfasına), ancak **fragment** URL'si burada sunucuya ulaşmaz.

Tarayıcı, bu fragment'i yerel olarak saklar ve daha sonraki adımlarda bu bilgiyi kullanabilir.

(E) Web-Hosted Client Resource'ın Yanıtı

Web barındırma istemcisi (örneğin, bir JavaScript uygulaması), verilen **redirection URI**'yi işler ve kullanıcının tarayıcısında saklanan fragment'i çıkararak içeriklere erişir:

- İstemci, **redirection URI**'yi tamamlar ve **fragment**'i çıkarır.
- Bu işlem genellikle bir **HTML sayfası** içinde yer alan bir script ile yapılır. Script, URL'deki fragment'i alır ve içindeki **access token**'ı (ve diğer parametreleri) çıkarır.

(F) Scriptin Çalıştırılması

Web-Hosted Client (örneğin, JavaScript uygulaması) tarafından sağlanan **script** (kod parçası) kullanılarak:

- **Access token**'ı, URL'den çıkarılır ve istemciye sağlanır.
- Kullanıcı aracı (tarayıcı), token'ı alır ve uygun işlemi yapmak üzere istemciye iletir.

Burada, scriptin kullanımı, token'ı almanın ve istemciye aktarmanın tek yoludur. Bu işlem, tamamen istemcinin JavaScript kodu içinde gerçekleşir.

(G) Access Token'ın İstemciye İletilmesi

Son adımda, kullanıcı aracı, çıkarılan **access token**'ı istemciye iletir. Bu token, istemcinin veri isteklerinde kimlik doğrulaması yapmak için kullanılır.

- İstemci, **access token**'ı aldıktan sonra, bu token'ı API çağrıları gibi işlemlerle kullanarak kaynağa erişim sağlamak için kullanabilir.
- Token, genellikle HTTP başlıkları içinde **Authorization** parametresi olarak iletilir:

Genel Özellikler

- **Implicit Grant** akışında, istemci ve kullanıcı arasında direkt bir etkileşim olduğu için, token'lar doğrudan kullanıcı aracısı aracılığıyla iletilir. Bu nedenle, güvenlik riskleri ve token sızıntısı gibi konularda dikkatli olunması gerekir.
- Bu akış genellikle **public clients** (örneğin, JavaScript tabanlı uygulamalar veya istemciler) için uygundur. Güvenli olmayan istemcilerde kullanılması, token'ların kolayca ele geçirilmesine neden olabilir.

Bu adımlar, OAuth 2.0 **Implicit Grant Flow** sürecinin nasıl işlediğini açık bir şekilde gösterir ve her bir adımda neler olduğunu anlamanızı sağlar.

4.2.1 Authorization Request (Yetkilendirme İsteği)

İstemci, yetkilendirme sunucusuna **access token** almak için bir yetkilendirme isteği gönderir. Bu istek, kullanıcıyı yetkilendirme sunucusuna yönlendirmek amacıyla bir URL'yi içerir. İstemci, **authorization request**'i başlatırken aşağıdaki parametreleri kullanarak yetkilendirme sunucusuna bir istek yapar.

Gerekli Parametreler

1. **response_type**

- **Zorunlu** bir parametredir. Bu parametre, istemcinin ne tür bir yanıt beklediğini belirtir. Implicit Grant flow için bu değer **"token"** olmalıdır, çünkü istemci doğrudan **access token** almak istiyor.
- Bu parametre **"token"** olarak ayarlanmışsa, istemci access token'ı doğrudan alacaktır. Bu, **Authorization Code Grant** türünde olmayan, daha basit ve daha hızlı bir erişim yöntemidir.

2. **client_id**
    - **Zorunlu** bir parametredir. Bu, istemcinin benzersiz kimliğini tanımlar. Genellikle istemci kayıt olduğunda verilen bir değerdir ve istemcinin, yetkilendirme sunucusunun tanıdığı uygulama olduğunu belirtir.
    - Örneğin, bir mobil uygulama veya bir web sitesi için istemci ID’si sağlanır.
3. **redirect_uri**
    - **Opsiyonel** bir parametredir. Bu parametre, yetkilendirme sunucusunun access token'ı gönderdiği URI’yi belirtir. Eğer bu parametre verilirse, yetkilendirme sunucusu yalnızca verilen URI'ye yönlendirme yapacaktır.
    - Eğer istemci, kaydederken belirli bir yönlendirme URI'si belirtilmişse, bu URI'nin gelen istekle uyuşması gerektiği için, **redirect_uri** parametresi doğrulanacaktır.
    - Eğer bu parametre sağlanmazsa, yetkilendirme sunucusu kaydedilen ilk URI'yi kullanacaktır.
4. **scope**
    - **Opsiyonel** bir parametredir. Bu, istemcinin erişmek istediği kaynakları veya izinleri tanımlar. Örneğin, kullanıcı bilgileri veya uygulama verileri gibi.
    - Bu parametre, istemcinin talep ettiği izinlerin genişliğini belirler. Eğer belirtilmezse, varsayılan bir kapsam kullanılabilir.
5. **state**
    - **Önerilen** bir parametredir. Bu parametre, istemcinin request ve callback (geri dönüş) arasında bir bağ kurmasına yardımcı olur. Özellikle, **Cross-Site Request Forgery (CSRF)** saldırılarına karşı koruma sağlamak için kullanılır.
    - **state** değeri, istemcinin başlattığı isteği tanımlayan ve yalnızca istemci tarafından bilinen bir değerdir. Bu parametreyi kullanarak, istemci istekleri ve geri dönüşleri eşleştirebilir. Yetkilendirme sunucusu, **state** parametresini de geri gönderir, böylece istemci, isteği ve yanıtı doğrulayabilir.

İstek Yapma (İstemci Tarafı)

Bu parametreler kullanılarak istemci, kullanıcının tarayıcısına yönlendirilir. Örneğin, istemci şu şekilde bir HTTP isteği gönderir:

GET /authorize?response_type=token&client_id=s6BhdRkqt3&state=xyz &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1 Host: server.example.com

Bu örnek, istemcinin yetkilendirme sunucusuna bir istek gönderdiğini gösterir. Burada:

- **response_type=token**: Erişim token'ı talep ediliyor.
- **client_id=s6BhdRkqt3**: İstemci kimliği.
- **state=xyz**: CSRF saldırılarına karşı koruma için istemcinin gönderdiği rastgele bir değer.
- **redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb**: Yönlendirme URI'si, yani başarılı bir yetkilendirmeden sonra kullanıcıyı geri gönderecek adres.

Yetkilendirme Sunucusunun İstek Doğrulaması

Yetkilendirme sunucusu, istemciden gelen bu isteği doğrular. **Doğrulama işlemi** şu adımları içerir:

1. **Tüm gerekli parametrelerin mevcut olup olmadığı kontrol edilir.**
2. **redirect_uri**: Yetkilendirme sunucusu, istemcinin kayıtlı olduğu yönlendirme URI'si ile gelen yönlendirme URI'sini karşılaştırır. Eğer bu iki URI birbirine uyuyorsa, istek geçerli kabul edilir. Eğer uyuşmazlık varsa, istek reddedilir.
3. **Diğer parametrelerin geçerliliği**: İstemcinin kimliği (client_id), yönlendirme URI'si ve scope gibi parametreler de doğrulanır.

Yetkilendirme Kararı ve Yönlendirme

Eğer istek geçerliyse ve istemci doğru parametrelerle başvurmuşsa, **yetkilendirme sunucusu**, kullanıcıyı yönlendirecek bir karar alır:

1. **Kullanıcıdan izin alma**: Yetkilendirme sunucusu, kullanıcının kimliğini doğrular ve kullanıcının erişim talebini onaylamasını ister.

2. **Erişim İzni**: Eğer kullanıcı erişimi onaylarsa, yetkilendirme sunucusu, kullanıcının tarayıcısını istenen **redirect_uri**'ye yönlendirir. Bu yönlendirme, URL'nin parçası olarak **access token** içerecek şekilde yapılır.

Bu süreç, istemcinin ve kullanıcının etkileşimi sırasında, istemcinin doğru erişim izinlerine sahip olup olmadığı ve kullanıcı tarafından onay alınıp alınmadığına bağlı olarak gerçekleşir.

Özet

Bu süreç, istemcinin yetkilendirme sunucusuna **access token** almak amacıyla yönlendirme yapması ve kullanıcıdan onay alması için kullanılan bir akıştır. Bu tür bir akış, genellikle **public clients** (örneğin, JavaScript tabanlı uygulamalar) için kullanılır çünkü istemci kimliği ve gizli anahtar gibi güvenlik bilgileri istemcinin tarafında saklanmaz.

4.2.2. Access Token Response

Yetkilendirme sunucusu, eğer kaynak sahibi (kullanıcı) erişim talebini onaylarsa, istemciye bir **access token** (erişim token'ı) gönderir. Bu token, istemcinin, belirli kaynaklara erişim sağlamasına olanak tanır. Bu token, **redirection URI**'nin **fragment** bölümüne eklenir ve istemciye gönderilir.

İşte bu yanıtın içerdiği parametreler ve açıklamaları:

Yanıt Parametreleri

1. **access_token**
    - **Zorunlu** bir parametredir. Yetkilendirme sunucusu, istemciye **erişim token'ı** verir. Bu token, istemcinin, kaynağa erişmesini sağlar.
    - **Erişim token'ı**, istemcinin sahip olduğu izinlere göre kaynaklara erişim sağlayan bir anahtar gibi çalışır. Token'ın içeriği genellikle şifreli veya imzalı olabilir.
2. **token_type**
    - **Zorunlu** bir parametredir. Bu, verilen token türünü belirtir. Bu değer genellikle **"bearer"** olur.
    - **"Bearer token"** kullanımı yaygındır, yani bu token'ı taşıyan her şey, ona erişim hakkına sahiptir. Örneğin, token'ı taşıyan bir istek, yetkilendirilmiş kabul edilir.
3. **expires_in**
    - **Önerilen** bir parametredir. Erişim token'ının **geçerlilik süresi** belirtilir. Bu parametre, token'ın ne kadar süreyle geçerli olduğunu belirler.
    - Örneğin, **"expires_in": 3600** değeri, token'ın 1 saat boyunca geçerli olacağı anlamına gelir.
    - Eğer bu parametre belirtilmezse, yetkilendirme sunucusu token'ın geçerliliği ile ilgili başka bir yöntem sağlayabilir veya varsayılan bir değer kullanabilir.
4. **scope**
    - **Opsiyonel** bir parametredir. Eğer erişim token'ı istemcinin talep ettiği kapsamla aynıysa, bu parametre gereksiz olabilir.
    - Ancak, eğer **access token** verilen kapsam, istemcinin talep ettiği kapsamla farklıysa, bu parametre **zorunlu** hale gelir.
    - Kapsam, token'ın hangi kaynaklara erişim sağladığını belirtir. Eğer kapsamla ilgili bir değişiklik varsa, bunu belirten parametre eklenir.
5. **state**
    - **Zorunlu** bir parametredir, ancak yalnızca istemci **state** parametresini talep ettiyse. Eğer istemci yetkilendirme isteğinde **state** parametresi göndermişse, yetkilendirme sunucusu bu değeri **geri gönderir**.
    - **State** parametresi, istemcinin yetkilendirme sürecinde tutmak istediği ve geri alması gereken bir değerdir. Bu, CSRF (Cross-Site Request Forgery) saldırılarını engellemeye yardımcı olur. İstemci, **state** parametresi ile talep ve yanıtları eşleştirerek güvenliği artırır.

Yönlendirme ve HTTP Yanıtı

Erişim token'ı yanıtı, yetkilendirme sunucusu tarafından kullanıcının tarayıcısına yönlendirme yapılacak şekilde gönderilir. Bu yanıt bir **HTTP 302 Found** yanıtı ile yapılır. Bu, istemciyi yeni bir URI'ye yönlendiren geçici bir HTTP durum kodudur. Yönlendirme, **Location** başlığı aracılığıyla yapılır ve erişim token'ı, **URI fragment** (örn., `#access_token=...`) içinde bulunur.

Örnek yanıt:

```
HTTP/1.1 302 Found
Location: http://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
             &state=xyz&token_type=example&expires_in=3600
```

Bu örnekte, yetkilendirme sunucusu kullanıcının tarayıcısını, istemcinin belirlediği yönlendirme URI'sine (bu örnekte **http://example.com/cb**) yönlendirir ve URL fragment'ine **access_token** ve diğer parametreleri ekler. Bu şekilde istemci, token'ı URI'den alır.

Önemli Notlar:

- **Fragment ve HTTP Yönlendirme**: Bazı tarayıcılar (user-agent'lar), **Location** başlığında fragment içeren bir yönlendirme yapmayı desteklemez. Bu durumda, istemciye başka yöntemlerle yönlendirme yapılması gerekebilir. Örneğin, istemci, HTML bir sayfa döndürebilir ve içinde yönlendirme yapılacak bir **'continue'** butonu ile kullanıcıyı yönlendirebilir.
- **Yanıt Parametreleri**: İstemci, tanımadığı parametreleri görmezden gelmelidir. Yalnızca **access_token**, **token_type**, **expires_in**, **scope**, ve **state** parametreleri ile ilgilenmelidir.
- **Access Token'ın Boyutu**: Token'ın boyutuna dair bir sınırlama yoktur, ancak istemci, token'ın boyutunun ne kadar olacağını bilmemelidir. Yetkilendirme sunucusu, token'ların boyutlarına dair bilgileri dökümante edebilir.

Özet

Access Token Response (Erişim Token'ı Yanıtı) sürecinde, yetkilendirme sunucusu, istemciye **access token**'ı, ilgili parametrelerle birlikte bir yönlendirme URI'si üzerinden iletir. Bu token, istemcinin kaynaklara erişim sağlamasına olanak tanır. Yanıt, ayrıca **token_type**, **expires_in**, **state** gibi parametreleri içerir.

4.2.2.1. Error Response

Error Response (Hata Yanıtı)

Hata yanıtı, istemcinin isteğiyle ilgili bir hata meydana geldiğinde yetkilendirme sunucusu tarafından gönderilir. İstemciden gelen yetkilendirme isteği geçerli değilse ya da kaynak sahibi isteği reddederse, yetkilendirme sunucusu, istemciyi geçerli olmayan yönlendirme URI'sine göndermemeli ve hatalı yanıtı **fragment** bölümüne ekleyerek istemciye iletmelidir.

Hata yanıtı, şu şekilde bir **HTTP 302 Found** yanıtı aracılığıyla yapılır ve yönlendirme URI'sinin fragment kısmında hata bilgileri bulunur. Bu fragment kısmı, istemcinin hatayı anlayabilmesi için gerekli olan bilgileri içerir.

Örnek hata yanıtı:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied&state=xyz
```

Bu örnekte, yetkilendirme sunucusu, istemciyi yönlendirme URI'sine gönderir (https://client.example.com/cb) ve hata bilgilerini **fragment** kısmında belirtir. Burada, hata kodu **access_denied** ve istemcinin gönderdiği **state** parametresi bulunmaktadır.

Hata Yanıtı Parametreleri

1. **error** (Zorunlu)
    - Bu parametre, hata durumunu tanımlar ve aşağıdaki hata kodlarından birini içerir:
    - **invalid_request**: İstek, eksik bir parametre içeriyor, geçersiz bir parametre değeri var, bir parametre birden fazla kez gönderilmiş veya istek yanlış formatta. Yani istemci isteği düzgün oluşturamamış.
    - **unauthorized_client**: İstemci, bu yöntemle erişim token'ı talep etme yetkisine sahip değil.
    - **access_denied**: Kaynak sahibi (kullanıcı) ya da yetkilendirme sunucusu, talebi reddetti.
    - **unsupported_response_type**: Yetkilendirme sunucusu, bu yöntemle erişim token'ı elde etmeyi desteklemiyor.
    - **invalid_scope**: Talep edilen kapsam geçersiz, bilinmiyor veya yanlış formatta.
    - **server_error**: Yetkilendirme sunucusu, beklenmeyen bir hata ile karşılaştı ve isteği yerine getiremedi. Bu hata kodu, sunucunun 500 Internal Server Error döndürmesinin engellendiği durumlarda gereklidir.
    - **temporarily_unavailable**: Yetkilendirme sunucusu, geçici bir aşırı yüklenme veya bakım nedeniyle isteği yerine getiremiyor. Bu, genellikle 503 Service Unavailable HTTP durum kodunun yerine kullanılacak hata kodudur.

    Önemli Not: **error** parametresi, sadece ASCII karakterlerden oluşmalıdır. Yani, karakterler **%x20-21**, **%x23-5B**, **%x5D-7E** arasında olmalıdır.

2. **error_description** (Opsiyonel)
    - İnsan tarafından okunabilir bir hata açıklaması sunar. Bu açıklama, hata hakkında daha fazla bilgi verir ve istemci geliştiricisinin hatayı anlamasına yardımcı olur.
    - Hata açıklamaları yalnızca ASCII karakterlerinden oluşmalıdır.
    - **error_description** parametresi genellikle istemciye hatanın daha detaylı bir açıklamasını sağlar, örneğin: "The redirect URI is not valid".
3. **error_uri** (Opsiyonel)
    - Hata hakkında daha fazla bilgi sağlayan bir **web URI**'si belirtir. Bu URI, istemci geliştiricisine hata hakkında ek bilgi veya açıklamalar sunar.
    - **error_uri** parametresi, bir **URI-reference** formatında olmalı ve yalnızca geçerli URI karakterlerini içermelidir.
4. **state** (Zorunlu)
    - Eğer istemci **state** parametresini yetkilendirme isteğinde belirtmişse, yetkilendirme sunucusu aynı değeri geri göndermelidir.
    - Bu parametre, istemcinin hata yanıtını doğru şekilde eşleştirmesini sağlar. **State** değeri, istemcinin yetkilendirme işlemi sırasında tutmak istediği özgün bir değeri içerir.

Hata Yanıtının iletilmesi

Yetkilendirme sunucusu, yukarıda belirtilen hata parametreleri ile birlikte bir **302 Found** HTTP yanıtı gönderir. Yanıtta, **Location** başlığında hata parametrelerinin bulunduğu yönlendirme URI'si verilir. Bu URI'de, **error** parametresi ile birlikte hata açıklamaları (varsa) ve hata URI'si de yer alır. Örneğin:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied&state=xyz
```

Önemli Notlar:

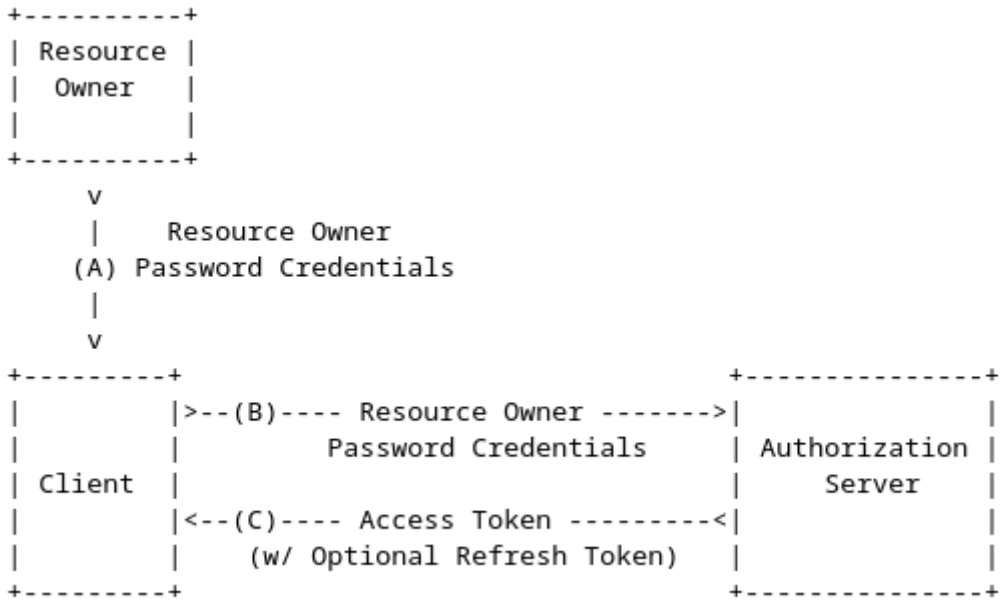
- **Yönlendirme URI'sinin Geçerliliği**: Yetkilendirme sunucusu, **redirection URI**'sinin geçersiz ya da hatalı olduğu durumlarda, otomatik olarak hatalı URI'ye yönlendirme yapmamalıdır. Bunun yerine, hata bilgileri içeren bir yanıt gönderir.
- **Hata Yanıtlarının Yapısı**: Hata yanıtlarında gönderilen parametreler ve URL fragment'inin doğru şekilde işlenmesi gerekir. Bu parametreler, istemcinin hatayı anlamasına ve durumu düzeltmesine yardımcı olacak bilgi sağlar.

Özet

Hata yanıtı, istemciden gelen geçersiz veya reddedilen yetkilendirme taleplerine karşılık yetkilendirme sunucusunun gönderdiği yanıttır. Bu yanıt, **error**, **error_description**, **error_uri**, ve **state** parametrelerini içerebilir ve istemcinin hata durumunu doğru bir şekilde işlemesi için gerekli bilgileri sağlar.

4.3. Resource Owner Password Credentials Grant (Kaynak Sahibi Şifre Kimlik Bilgisi Yetkilendirme Akışı), OAuth 2.0 protokolünün bir yetkilendirme türüdür ve belirli durumlarda kullanılır. Bu akış, kaynak sahibinin (kullanıcının) doğrudan uygulama ile güven ilişkisi kurduğu senaryolarda kullanılır, örneğin cihaz işletim sistemleri veya yüksek yetkili uygulamalar. Bu akış, kullanıcının kullanıcı adı ve şifresini (genellikle etkileşimli bir form aracılığıyla) uygulamaya vererek doğrulama yapmasına olanak tanır. Bununla birlikte, bu akış yalnızca diğer yetkilendirme akışlarının uygulanabilir olmadığı durumlarda kullanılmalıdır, çünkü güvenlik riskleri içerebilir.

Kaynak Sahibi Şifre Kimlik Bilgisi Yetkilendirme Akışı (Password Credentials Grant)



Akışın Detayları:

1. **(A) Kaynak Sahibi (User)**, istemciye kullanıcı adı ve şifresini sağlar. Burada, kullanıcı şifre bilgilerini doğrudan istemci uygulamasına verir.
2. **(B) İstemci**, kaynak sahibinden aldığı kullanıcı adı ve şifreyi, yetkilendirme sunucusunun **token endpoint**'ine gönderir. Bu aşamada istemci, yetkilendirme sunucusu ile kimlik doğrulaması yapar. İstemci, genellikle **client_id** ve **client_secret** gibi bilgileri kullanarak yetkilendirme sunucusuna kimlik doğrulama bilgilerini iletir.
3. **(C) Yetkilendirme Sunucusu**, istemcinin kimlik bilgilerini doğrular ve kaynak sahibinin (kullanıcının) sağladığı şifre bilgilerini de kontrol eder. Eğer kullanıcı adı ve şifre doğruysa, yetkilendirme sunucusu bir **access token** (erişim token'ı) ve opsiyonel olarak bir **refresh token** (yenileme token'ı) oluşturur ve istemciye gönderir.

Şekil 5: Kaynak Sahibi Şifre Kimlik Bilgisi Akışı

Bu akış, kaynak sahibinin kullanıcı adı ve şifreyi doğrudan istemciye vererek, istemcinin yetkilendirme sunucusundan bir erişim token'ı almasını sağlar. Bu akış, genellikle aşağıdaki senaryolarda kullanılır:

- **Güvenli İstemciler İçin**: Kaynak sahibi ve istemci arasında güven ilişkisi olduğunda, örneğin bir cihazın işletim sistemi veya güvenli bir uygulama.
- **Mevcut Kimlik Doğrulama Yöntemlerinin OAuth'a Taşınması**: Bu akış, mevcut kimlik doğrulama yöntemlerini OAuth 2.0 protokolüne uyarlamak için de kullanılabilir. Örneğin, HTTP Basic veya Digest kimlik doğrulaması kullanan eski sistemlerden OAuth'a geçiş yapılabilir.

Dikkat Edilmesi Gerekenler:

- **Güvenlik Riski**: Kaynak sahibinin kullanıcı adı ve şifresini doğrudan istemciye vermesi gerektiği için, bu akış çok dikkatli kullanılmalıdır. Kaynak sahibinin şifre bilgileri istemcinin kontrolüne geçer, bu nedenle güvenliğin sağlanması önemlidir. Ayrıca, istemcinin bu bilgileri güvenli bir şekilde iletmesi ve saklaması gerekir.
- **Alternatif Akışlar**: Bu akış, diğer akışlar (örneğin, Authorization Code Grant veya Implicit Grant) uygun olmadığında kullanılmalıdır. Diğer akışlar daha güvenlidir, çünkü kullanıcı adı ve şifre doğrudan istemci ile paylaşılmaz.

Örnek Durum:

Bir mobil uygulama, kullanıcı adı ve şifre bilgilerini alıp yetkilendirme sunucusuna göndererek bir erişim token'ı alır. Bu token, uygulamanın kullanıcının adına API'lere erişmesini sağlar.

Akışın Kullanıldığı Durumlar:

- **Herkesin Güvendiği Uygulamalar**: Cihazlar veya yüksek güvenlik gereksinimleri olan uygulamalar, örneğin bir mobil uygulama ya da masaüstü uygulaması, bu akışı kullanabilir.
- **Eski Yöntemlerin OAuth'a Entegre Edilmesi**: Mevcut kimlik doğrulama yöntemlerinin OAuth ile entegre edilmesi gerektiğinde, bu akış kullanılabilir.

Sonuç:

Kaynak Sahibi Şifre Kimlik Bilgisi Yetkilendirme Akışı, güvenli ve güvenilir bir ortamda kullanılmalıdır. Yalnızca diğer yöntemlerin geçerli olmadığı durumlar için önerilir. Bu akışın kullanılabilmesi için istemci ve kaynak sahibi arasında güvenli bir ilişki olması gerekir.

4.3.1. Yetkilendirme İsteği ve Yanıtı

İstemcinin kaynak sahibi kimlik bilgilerini elde etme yöntemi, bu spesifikasyonun kapsamı dışındadır. İstemci, bir erişim token'ı alındıktan sonra kimlik bilgilerini imha etmelidir.

4.3.2. Erişim Token'ı İsteği

İstemci, "application/x-www-form-urlencoded" formatında, UTF-8 karakter kodlaması ile HTTP istek gövdesine ekleyerek token uç noktasına aşağıdaki parametreleri içeren bir istek gönderir:

- **grant_type**: **ZORUNLU**. Değeri "password" olarak ayarlanmalıdır.
- **username**: **ZORUNLU**. Kaynak sahibinin kullanıcı adı.
- **password**: **ZORUNLU**. Kaynak sahibinin şifresi.
- **scope**: **İSTEĞE BAĞLI**. Erişim isteğinin kapsamı, Bölüm 3.3'te açıklandığı gibi.

Eğer istemci türü gizli (confidential) ise ya da istemciye kimlik doğrulama gereksinimleri verilmişse (veya istemci kimlik bilgileri verilmişse), istemci, **Bölüm 3.2.1**'de açıklandığı şekilde yetkilendirme sunucusuyla kimlik doğrulaması yapmalıdır.

Örneğin, istemci aşağıdaki HTTP isteğini (sadece görsel amaçlı satır sonları eklenmiş) TLS (Transport Layer Security) üzerinden gönderir:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w
```

Yetkilendirme sunucusu şu adımları atmalıdır:

- Gizli istemciler için ya da kimlik bilgisi verilen istemciler (veya başka kimlik doğrulama gereksinimleri olanlar) için istemci kimlik doğrulamasını zorunlu tutmalıdır.
- İstemci kimlik doğrulaması yapılmışsa, istemciyi doğrulamalıdır.
- Kaynak sahibi şifre bilgilerini mevcut şifre doğrulama algoritmasıyla doğrulamalıdır.

Bu erişim token'ı isteği, kaynak sahibinin şifresini kullandığı için, yetkilendirme sunucusunun uç noktayı brute force saldırılarına karşı koruması gerekmektedir (örneğin, hız sınırlaması kullanmak veya uyarılar üretmek gibi önlemler alarak).

4.3.3. Erişim Token'ı Yanıtı

Eğer erişim token'ı isteği geçerli ve yetkilendirilmişse, yetkilendirme sunucusu, **Bölüm 5.1**'de açıklandığı gibi bir erişim token'ı ve isteğe bağlı bir yenileme token'ı (refresh token) verir. Eğer istek istemci kimlik doğrulamasını başaramazsa ya da geçersizse, yetkilendirme sunucusu **Bölüm 5.2**'de açıklandığı gibi bir hata yanıtı döndürür.

Başarılı bir yanıt örneği:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "example",
    "expires_in": 3600,
    "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
    "example_parameter": "example_value"
}
```

Bu yanıtın içerdiği parametreler:

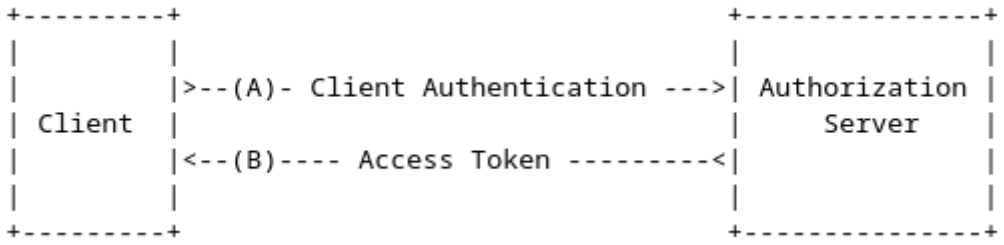
- **access_token**: Erişim token'ı, istemcinin kaynaklara erişmesini sağlayacak olan anahtar.
- **token_type**: Token türü, burada "example" değeri bir örnektir, uygulamada genellikle "bearer" kullanılır.
- **expires_in**: Erişim token'ının geçerlilik süresi, saniye cinsinden belirtilir (bu örnekte 3600 saniye = 1 saat).
- **refresh_token**: İsteğe bağlı bir yenileme token'ı, erişim token'ı süresi dolduğunda yeni bir erişim token'ı almak için kullanılabilir.
- **example_parameter**: Yanıtla birlikte döndürülen ek bir parametre, bu genellikle yetkilendirme sunucusu tarafından eklenen özel bir parametre olabilir.

Eğer istek başarılıysa, istemci bu bilgileri alır ve uygun şekilde erişim sağlamak için kullanabilir.

4.4. Client Credentials Grant

Client Credentials Grant türü, istemcinin sadece kendi kimlik bilgilerini kullanarak erişim token'ı talep ettiği bir OAuth2.0 akışıdır. Bu akış, istemcinin **kendi kontrolündeki** korunan kaynaklara veya **daha önce yetkilendirme sunucusuyla düzenlenmiş** başka bir kaynak sahibinin kaynaklarına erişmek için kullanılır. Bu türde, istemci kimlik doğrulaması yaparak doğrudan kaynaklarına erişim sağlamak amacıyla bir erişim token'ı talep eder.

Bu grant tipi yalnızca **gizli istemciler (confidential clients)** tarafından kullanılabilir. Gizli istemciler, istemci kimlik bilgilerini güvenli bir şekilde saklayabilen ve kimlik doğrulamasını güvenli bir şekilde yapabilen uygulamalardır.



Akışın adımları (Şekil 6'da gösterildiği gibi):

1. **(A) İstemci Kimlik Doğrulaması:**
   İstemci, yetkilendirme sunucusu ile kimliğini doğrular ve token endpoint'ine bir erişim token'ı talebi gönderir. Bu aşamada, istemci kimlik bilgilerini kullanarak sunucuya başvurur.
2. **(B) Yetkilendirme Sunucusunun Kimlik Doğrulaması:**
   Yetkilendirme sunucusu, istemcinin kimlik bilgilerini doğrular. Eğer kimlik doğrulaması geçerli ise, sunucu istemciye bir erişim token'ı (access token) gönderir.

Bu flow, genellikle arka planda çalışan uygulamalar veya istemcilerin kendi kaynaklarına erişim sağlamak için kullanılır. Örneğin, bir API'nin yalnızca uygulama tarafından erişilmesi gerektiğinde, istemci kimlik bilgileriyle yapılan bu akış uygun olur.

Özetle:

- İstemci, yalnızca kendi kimlik bilgileriyle yetkilendirme sunucusuna başvurur.
- Yetkilendirme sunucusu, istemcinin kimlik bilgilerini doğrular ve geçerli ise bir erişim token'ı gönderir.

4.4.1. Yetkilendirme Talebi ve Yanıtı

İstemci kimlik doğrulaması, yetkilendirme grant'ı (izin verme) olarak kullanıldığından, ek bir yetkilendirme talebine gerek yoktur.

4.4.2. Erişim Belirteci Talebi

İstemci, aşağıdaki parametreleri "application/x-www-form-urlencoded" formatında, Ek B'ye göre ve HTTP isteği varlık gövdesinde UTF-8 karakter kodlaması kullanarak token uç noktasına bir talep gönderir:

- **grant_type**
  ZORUNLU. Değer "client_credentials" olarak ayarlanmalıdır.
- **scope**
  İSTEĞE BAĞLI. Erişim talebinin kapsamı, Bölüm 3.3'te açıklandığı gibi.

İstemci, Bölüm 3.2.1'de açıklandığı şekilde yetkilendirme sunucusuyla kimlik doğrulaması yapmalıdır.

Örneğin, istemci aşağıdaki HTTP isteğini iletir (sadece gösterim amacıyla ekstra satır aralıkları eklenmiştir):

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

Yetkilendirme sunucusu, istemciyi kimlik doğrulamalıdır.

4.4.3. Erişim Belirteci Yanıtı

Erişim belirteci talebi geçerli ve yetkilendirilmişse, yetkilendirme sunucusu, Bölüm 5.1'de açıklandığı gibi bir erişim belirteci verir. Bir yenileme belirteci **DAHİL EDİLMEMELİDİR**. Eğer talep, istemci kimlik doğrulaması hatası nedeniyle başarısız olduysa veya geçersizse, yetkilendirme sunucusu, Bölüm 5.2'de açıklandığı gibi bir hata yanıtı döndürür.

Başarılı bir yanıt örneği:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
	"access_token":"2YotnFZFEjr1zCsicMWpAA",
	"token_type":"example",
	"expires_in":3600,
	"example_parameter":"example_value"
}
```

4.5. Extension grant (uzantı yetkilendirme türü), OAuth 2.0'ın standart yetkilendirme türlerinden farklı bir şekilde çalışır. Bu tür, istemcinin yetkilendirme sunucusuna özel bir yetkilendirme türünü belirtmesini sağlar. Uzantı grant türleri, belirli bir işlevselliği sağlamak için genişletilebilir ve bu türlerin nasıl kullanılacağı, yetkilendirme sunucusunun tanımladığı mutlak bir URI (Uniform Resource Identifier) ile belirlenir.

Adımlar ve Açıklama:

1. **Grant Type Belirleme**:
   İstemci, token talep ederken, "grant_type" parametresini kullanarak uzantı türünü belirtir. Bu tür, yetkilendirme sunucusu tarafından tanımlanmış bir URI ile belirlenir. Yani, her uzantı türü için yetkilendirme sunucusu tarafından tanımlanan özel bir URI kullanılır. Örneğin, **SAML 2.0** kullanarak erişim belirteci almak için bir URI belirlenmiştir ve bu URI, istemci tarafından kullanılır.

2. **Erişim Belirteci Talebi**:
   İstemci, bu grant türünü kullanarak erişim belirteci almak için bir HTTP isteği gönderir. Bu istekte, grant türünü (örn. `grant_type=urn:ietf:params:oauth:grant-type:saml2-bearer`) ve gereken diğer parametreleri (örneğin, SAML beyanı) ekler.

3. **Geçerli ve Yetkilendirilmiş İstek**:
   Eğer istemcinin gönderdiği istek geçerli ve yetkilendirilmişse, yetkilendirme sunucusu, istemciye bir erişim belirteci ve isteğe bağlı bir yenileme belirteci gönderir.

4. **Hatalı veya Geçersiz İstek**:
   Eğer istek geçersizse veya istemci kimlik doğrulaması başarısız olursa, yetkilendirme sunucusu hata yanıtı döndürür.

Örnek:

Buradaki örnekte, istemci **SAML 2.0** kullanarak bir erişim belirteci almak istiyor. Bunun için, aşağıdaki gibi bir HTTP isteği gönderiyor:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-
bearer&assertion=PEFzc2VydGlvbiBJc3N1ZUluc3RhbnQ9IjIwMTEtMDU
```

Burada, `grant_type` parametresi belirli bir uzantıyı (`saml2-bearer`) belirtiyor ve `assertion` parametresi ise SAML beyanını taşıyor.

Eğer bu istek başarılı olursa, yetkilendirme sunucusu erişim belirteci ve yenileme belirteci gönderir. Eğer geçersizse, hata mesajı döner.

Bu, OAuth 2.0'ın esnekliğinden faydalanarak, özel yetkilendirme türlerinin eklenmesine olanak tanır.

5. Issuing an Access Token

Eğer erişim belirteci isteği geçerli ve yetkilendirilmişse, yetkilendirme sunucusu **Bölüm 5.1'de** açıklandığı şekilde bir erişim belirteci ve isteğe bağlı olarak bir yenileme belirteci oluşturur. Eğer istek istemci kimlik doğrulamasında başarısız olursa veya geçersizse, yetkilendirme sunucusu **Bölüm 5.2'de** açıklandığı şekilde bir hata yanıtı döner.

`5.1. Successful Response`

Bu bölüm, başarılı bir erişim belirteci yanıtının nasıl oluşturulduğunu ve neleri içerdiğini detaylandırır. İşte açıklaması:

Erişim Belirteci ve Yenileme Belirteci

Yetkilendirme sunucusu, başarılı bir istek sonrasında şu bilgileri içeren bir HTTP yanıtı oluşturur:

1. **access_token (Gerekli):**
    - Yetkilendirme sunucusu tarafından verilen erişim belirtecidir. Bu belirteç, API gibi korunan kaynaklara erişimde kullanılır.
2. **token_type (Gerekli):**
    - Verilen belirtecin türüdür (örneğin, "Bearer"). Bu, **Bölüm 7.1'de** açıklanmıştır ve büyük/küçük harfe duyarsızdır.
3. **expires_in (Tavsiye Edilir):**
    - Erişim belirtecinin geçerlilik süresini (saniye olarak) belirtir. Örneğin, `3600` değeri, erişim belirtecinin bir saat geçerli olacağını gösterir. Bu parametre sağlanmazsa, yetkilendirme sunucusu başka yollarla son kullanma bilgisini sağlamalıdır veya varsayılan bir süreyi dokümante etmelidir.
4. **refresh_token (Opsiyonel):**
    - Yenileme belirteci, mevcut erişim belirtecinin süresi dolduğunda yeni bir belirteç almak için kullanılır. Bu işlem, **Bölüm 6'da** açıklanmıştır.
5. **scope (Opsiyonel):**
    - Eğer istemci tarafından talep edilen kapsamla aynıysa bu parametreye gerek yoktur. Farklıysa bu parametre sağlanmalıdır. Bu, erişim belirtecinin hangi izinlere sahip olduğunu belirtir.

JSON Formatı

Yetkilendirme sunucusu, bu parametreleri **JSON** formatında HTTP yanıt gövdesine dahil eder.

- Parametreler, JSON yapısında en üst seviyede yer alır.
- İsimler ve metin değerleri, JSON dizesi (string) olarak kodlanır.
- Sayısal değerler, JSON sayıları olarak yer alır.
- Parametrelerin sıralaması önemli değildir ve değişebilir.

Örnek Yanıt

Aşağıdaki yanıt, başarılı bir erişim belirteci isteğine örnek gösterir:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter": "example_value"
}
```

Güvenlik Önlemleri

Yetkilendirme sunucusu, güvenlik için şu başlıkları her yanıtında içermelidir:

1. **Cache-Control:**
    - `no-store` değeri ile yanıtın tarayıcı veya ara önbellekler tarafından depolanmasını engeller.
2. **Pragma:**
    - `no-cache` değeri ile önceki önbelleğe alınmış verilerin kullanılmasını engeller.

İstemciye Yönelik Notlar

- İstemci, yanıtta tanımadığı parametre adlarını görmezden gelmelidir.
- Erişim belirteci veya diğer parametrelerin boyutu tanımlanmamıştır; bu nedenle istemci, belirli bir boyut beklentisiyle hareket etmemelidir.
- Yetkilendirme sunucusu, verdiği parametrelerin boyutunu dokümante etmelidir.

Bu süreç, güvenliğin sağlanması ve istemcilerin yetkilendirme sunucusundan aldıkları belirteçleri doğru şekilde işlemesi için önemlidir.

5.2. Error Response

Bu bölüm, erişim belirteci isteği sırasında oluşabilecek hatalar ve bu hatalar karşısında yetkilendirme sunucusunun nasıl bir yanıt oluşturması gerektiğini açıklar.

Hata Yanıtının Temelleri

- Yetkilendirme sunucusu, bir hata durumunda genellikle **HTTP 400 (Bad Request)** durum kodu döner (aksi belirtilmedikçe).
- Yanıt, aşağıdaki parametreleri içerir ve JSON formatında iletilir.

Hata Parametreleri

1. **error (Gerekli):**
    - Hatanın türünü belirten bir ASCII hata kodu içerir. Bu kodlar aşağıdaki gibidir:
        - **invalid_request:**
            - İstek aşağıdaki nedenlerden dolayı geçersizdir:
                - Gerekli bir parametre eksik.
                - Desteklenmeyen bir parametre değeri kullanılmış.
                - Parametre tekrar edilmiş.
                - Birden fazla kimlik bilgisi veya kimlik doğrulama mekanizması kullanılmış.
                - İstek başka bir şekilde hatalı oluşturulmuş.
        - **invalid_client:**
            - İstemcinin kimlik doğrulaması başarısız olmuştur. Örneğin:
                - İstemci tanınmıyor.
                - İstemci kimlik doğrulama bilgileri sağlanmamış.
                - Desteklenmeyen bir kimlik doğrulama yöntemi kullanılmış.
            - Yetkilendirme sunucusu, **HTTP 401 (Unauthorized)** durum kodu dönebilir ve desteklenen kimlik doğrulama şemalarını belirtmek için `WWW-Authenticate` başlığını içermelidir.
        - **invalid_grant:**
            - Sağlanan yetkilendirme yetkisi (örneğin, yetkilendirme kodu, kaynak sahibi kimlik bilgileri) veya yenileme belirteci:
                - Geçersiz, süresi dolmuş, iptal edilmiş.
                - Yetkilendirme isteğinde kullanılan yönlendirme URI'siyle eşleşmiyor.
                - Başka bir istemci için verilmiş.
        - **unauthorized_client:**
            - Kimliği doğrulanmış istemci, bu yetkilendirme yetkisini kullanmaya yetkili değil.
        - **unsupported_grant_type:**
            - Kullanılan yetkilendirme türü, yetkilendirme sunucusu tarafından desteklenmiyor.
        - **invalid_scope:**
            - Talep edilen kapsam:
                - Geçersiz, bilinmiyor, hatalı biçimlendirilmiş.
                - Kaynak sahibinin verdiği kapsamı aşıyor.
2. **error_description (Opsiyonel):**
    - İnsan tarafından okunabilir bir açıklama sağlar.
    - Hatanın neden meydana geldiğini anlaması için istemci geliştiricisine yardımcı olur.

3. **error_uri (Opsiyonel):**
    - Hata hakkında daha fazla bilgi içeren, okunabilir bir web sayfasını tanımlayan URI'yi içerir.

JSON Formatı

- Tüm parametreler, JSON formatında yanıt gövdesine eklenir.
- Parametre isimleri ve değerleri JSON dizesi olarak ifade edilir.
- Parametrelerin sıralaması önemli değildir.
- Örnek hata yanıtı:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "error": "invalid_request"
}
```

Güvenlik Notları

- Yetkilendirme sunucusu, yanıtlarında aşağıdaki başlıkları içermelidir:
    - **Cache-Control:** `no-store` ile duyarlı bilgilerin önbelleğe alınmasını engeller.
    - **Pragma:** `no-cache` ile önbellekten yanlışlıkla bilgi alınmasını önler.

İstemciye Öneriler

- İstemci, tanımadığı `error` parametresi değerlerini yok saymalıdır.
- Hata mesajları, istemci geliştiricisinin hatayı anlamasına ve çözmesine yardımcı olacak şekilde tasarlanmalıdır.

6. Refreshing an Access Token

Bu bölüm, bir istemcinin mevcut bir erişim belirtecinin süresi dolduğunda veya yenilenmesi gerektiğinde, **refresh token** kullanarak yeni bir erişim belirteci talep etme sürecini açıklamaktadır.

Yenileme Talebi (Refresh Request)

İstemci, yeni bir erişim belirteci almak için bir **refresh token** kullanır ve aşağıdaki parametreleri içeren bir HTTP isteği yapar. İstek, `application/x-www-form-urlencoded` formatında ve **UTF-8** karakter kodlamasıyla gönderilir.

Talep Parametreleri

1. **grant_type (Zorunlu):**
    - Değeri **"refresh_token"** olarak ayarlanmalıdır.
    - Bu, talebin bir yenileme talebi olduğunu belirtir.
2. **refresh_token (Zorunlu):**
    - Daha önce istemciye verilen refresh token.
    - Bu, yeni bir erişim belirteci almak için kullanılır.
3. **scope (Opsiyonel):**
    - Talep edilen erişim kapsamı (isteğe bağlıdır).
    - Kaynak sahibinin başlangıçta verdiği kapsam dışında bir kapsam talep edilemez.
    - Eğer belirtilmezse, başlangıçta verilen kapsam varsayılan olarak kullanılır.

Refresh Token Güvenliği

- **Refresh token**, uzun süre geçerli olan bir kimlik bilgisi olduğu için, yalnızca token'ı alan istemciyle bağlantılıdır.
- İstemci tipi "confidential" ise veya istemciye kimlik doğrulama bilgileri verildiyse (örneğin istemci kimlik bilgileri), istemci yetkilendirme sunucusuyla kimlik doğrulaması yapmalıdır.

Örnek HTTP Yenileme Talebi

Aşağıda, bir istemcinin güvenli bir bağlantı üzerinden yaptığı örnek yenileme talebi verilmiştir:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

Detaylar:

- **Authorization Header:**
    - İstemci kimlik doğrulama bilgilerini içerir (örneğin, Base64 ile kodlanmış istemci kimliği ve istemci sırrı).
- **grant_type ve refresh_token:** Talebin içeriğinde gönderilir.

Yetkilendirme Sunucusunun Sorumlulukları

Yetkilendirme sunucusu, yenileme talebi sırasında aşağıdaki adımları takip etmelidir:

1. **İstemci Kimlik Doğrulaması:**
    - Confidential istemciler ve kimlik bilgileri verilen istemciler için kimlik doğrulama yapılmalıdır.
    - Refresh token'ın doğrulanan istemciye ait olduğundan emin olunmalıdır.
2. **Refresh Token Doğrulaması:**
    - Sağlanan refresh token geçerli ve yetkilendirilmiş olmalıdır.
3. **Erişim Belirtecini Yenileme:**
    - Eğer talep geçerliyse, istemciye yeni bir erişim belirteci verilmelidir.

Yeni Refresh Token Verme Seçeneği

- Yetkilendirme sunucusu, istemciye yeni bir refresh token sağlayabilir.
    - **Yeni refresh token verilirse:** İstemci eski refresh token'ı iptal etmeli ve yeni token'ı kullanmalıdır.
    - **Eski refresh token iptali:** Yeni bir refresh token sağlanırsa, sunucu eski refresh token'ı iptal edebilir.
- **Yeni Token'ın Kapsamı:**
    - Yeni refresh token verilirse, istemci tarafından sağlanan refresh token'ın kapsamı ile aynı olmalıdır.

Hata Durumları

Eğer:

- Talep doğrulama başarısız olursa,
- Refresh token geçersizse,
- İstemci kimlik doğrulaması yapılamazsa, **yetkilendirme sunucusu**, bir hata yanıtı döner .

Özet

Refresh token, istemcinin kaynak sahibinin yeniden müdahalesine gerek kalmadan yeni bir erişim belirteci talep etmesini sağlar. Bu süreç, güvenlik açısından dikkatle yönetilmeli, istemci kimlik

doğrulaması ve refresh token doğrulaması kesinlikle yapılmalıdır. Sunucu, refresh token'ın yanlış kullanılmasını önlemek için belirli durumlarda yeni bir refresh token verme ve eski token'ı iptal etme mekanizması sağlayabilir.

7. Accessing Protected Resources

İstemci, **erişim belirteci**ni kaynak sunucusuna sunarak korunan kaynaklara erişir. **Kaynak sunucusu**, erişim belirtecini doğrulamalı, süresinin dolmadığından emin olmalı ve belirtecin kapsamının talep edilen kaynağı kapsayıp kapsamadığını kontrol etmelidir.

Kaynak sunucusunun, erişim belirtecini doğrulamak için kullandığı yöntemler (ve herhangi bir hata yanıtı) bu spesifikasyonun kapsamı dışında kalır. Ancak, genellikle kaynak sunucusu ile yetkilendirme sunucusu arasında bir etkileşim veya koordinasyon içerir.

İstemcinin, kaynak sunucusuyla kimlik doğrulaması yapmak için erişim belirtecini nasıl kullandığı, yetkilendirme sunucusu tarafından verilen erişim belirteci türüne bağlıdır. Genellikle, istemci, [RFC2617] ile tanımlanan HTTP **"Authorization"** başlık alanını kullanır ve bu başlık, kullanılan erişim belirteci türünün spesifikasyonu tarafından tanımlanan bir kimlik doğrulama şeması içerir (örneğin, [RFC6750]).

7.1. Access Token Types

Erişim belirteci türü, istemcinin korunan bir kaynak talebi yaparken erişim belirtecini nasıl kullanacağını ve bunun için gerekli bilgileri sağlar (türle ilgili özel nitelikler dahil). **İstemci**, belirteç türünü anlamıyorsa erişim belirtecini kullanmamalıdır.

Örnekler:

1. **Bearer Token**
   [RFC6750]'de tanımlanan "bearer" belirteç türü, erişim belirteci dizesinin talebe doğrudan eklenmesiyle kullanılır:

   ```
   GET /resource/1 HTTP/1.1
   Host: example.com
   Authorization: Bearer mF_9.B5f-4.1JqM
   ```

   Bu, istemcinin belirteci taşıdığı ve sunucuya gönderdiği basit bir yöntemdir.

2. **MAC Token**
   [OAuth-HTTP-MAC] tarafından tanımlanan "mac" belirteç türü, erişim belirteci ile birlikte bir **Mesaj Kimlik Doğrulama Kodu (MAC)** anahtarı sağlanarak kullanılır. Bu anahtar, HTTP isteklerinin belirli bileşenlerini imzalamak için kullanılır:

   ```
   GET /resource/1 HTTP/1.1
   Host: example.com
   Authorization: MAC id="h480djs93hd8",
                      nonce="274312:dj83hs9s",
                      mac="kDZvddkndxvhGRXZhvuDjEWhGeE="
   ```

   Bu yöntem, daha fazla güvenlik gerektiren senaryolarda tercih edilir.

Geliştiriciler İçin Not:

Bu örnekler yalnızca açıklayıcı amaçlarla verilmiştir. Geliştiriciler, gerçek projelerinde **[RFC6750]** ve **[OAuth-HTTP-MAC]** spesifikasyonlarını inceleyerek kullanımlarını doğru bir şekilde uygulamalıdır.

Erişim Belirteci Türleri ve Özellikleri:

Her erişim belirteci türü, istemciye **"access_token"** yanıt parametresiyle birlikte gönderilen ek nitelikleri (varsa) tanımlar. Ayrıca, korunan bir kaynak talebi yapılırken erişim belirtecinin HTTP kimlik doğrulama yöntemiyle nasıl dahil edileceğini belirtir.

Bu, farklı erişim belirteci türlerinin kullanımını kolaylaştırmak için yapılan bir standartlaşmayı ifade eder.

`7.2. Error Response`

Bu bölüm, OAuth ile korunan bir kaynağa erişim talebinin başarısız olması durumunda kaynak sunucusunun istemciye hata bildiriminde bulunma yollarını ele alır. Belirli hata yanıtlarının detayları bu spesifikasyonun kapsamına girmese de, bu doküman OAuth belirteç doğrulama mekanizmalarında kullanılacak hata değerlerini yönetmek için ortak bir kayıt sistemi tanımlamaktadır (Bölüm 11.4).

Açıklama:

1. **Hata Bildirimi:**
    - Kaynak sunucusu, erişim talebinin başarısız olması durumunda istemciye hata hakkında bilgi vermelidir.
    - Spesifik hata yanıtlarının detayları bu dokümanda belirtilmemiştir.
    - Ancak, OAuth için hata değerleri arasında ortak bir standardizasyon sağlamak amacıyla bir hata değeri kaydı oluşturulmuştur.
2. **Yeni Kimlik Doğrulama Mekanizmaları:**
    - Primarily (öncelikli olarak) OAuth belirteç doğrulaması için tasarlanan yeni kimlik doğrulama mekanizmaları:
        - İstemciye hata durumu ile ilgili bilgi sağlamak için bir mekanizma tanımlamalıdır.
        - Bu mekanizmalar, geçerli hata kodlarını kayıtlı değerlerin bir alt kümesi ile sınırlayabilir.
        - Hata kodunun bir parametre aracılığıyla iletildiği durumlarda, parametre adı genellikle **"error"** olmalıdır.
3. **Diğer Kimlik Doğrulama Mekanizmaları:**
    - OAuth belirteç doğrulaması için kullanılabilen, ancak bu amaçla öncelikli olarak tasarlanmamış diğer mekanizmalar:
        - Kendi hata değerlerini bu kayıtla ilişkilendirebilir.
        - Ancak bunu yapmak zorunda değillerdir.

4. **Ek Hata Parametreleri:**
    - Yeni kimlik doğrulama mekanizmaları, istemciye daha açıklayıcı hata bilgisi sağlamak için şu parametreleri kullanabilir:
        - **"error_description"**: Hata hakkında insan tarafından okunabilir bir açıklama sağlar.
        - **"error_uri"**: Hata ile ilgili daha fazla bilgiye ulaşılabilecek bir URI sağlar.
    - Bu parametreler, bu dokümandaki kullanımlarıyla paralel bir şekilde kullanılabilir.

Özet:

Bu bölüm, OAuth belirteç doğrulaması için kullanılan hata değerlerinin standardize edilmesini ve yeni kimlik doğrulama mekanizmalarının bu standarda uygun olarak tasarlanmasını önerir. Bu sayede, istemciler hata durumlarını daha kolay anlayabilir ve ele alabilir. Ek olarak, daha karmaşık senaryolarda detaylı hata bilgisi sağlamak için ek parametreler kullanılabilir.

8.1. Access Token Türlerinin Tanımlanması

1. **Kayıt Yöntemleri:**
    - Belirteç türleri, iki şekilde tanımlanabilir:
        - **Access Token Types** kayıt defterine kayıt edilerek (Bölüm 11.1'deki prosedürlere göre).
        - Benzersiz bir **mutlak URI** adı kullanılarak.
2. **Kısıtlamalar:**
    - URI tabanlı adlandırma sadece belirli, vendor (satıcı)-odaklı ve genellikle başka sistemlere uygulanabilir olmayan durumlarla sınırlı olmalıdır.
    - Diğer tüm türler **kayıtlı olmalıdır** ve **type-name ABNF** kurallarına uymalıdır.
3. **Tip Adlandırma Kuralları:**
    - Yeni bir HTTP kimlik doğrulama şeması tanımlanıyorsa, belirteç türü adı, HTTP kimlik doğrulama şeması adıyla aynı olmalıdır.
    - Örnek olarak, "example" belirteç türü, sadece örneklerde kullanılmak üzere ayrılmıştır.

8.2. Yeni Uç Nokta Parametrelerinin Tanımlanması

1. **Kayıt Gereklilikleri:**
    - Yeni parametreler, OAuth Parameters kayıt defterine kayıt edilmelidir (Bölüm 11.2).
2. **Adlandırma Kuralları:**
    - Parametre adları, **param-name ABNF** kurallarına uymalıdır.
    - Parametre değerlerinin sentaksı iyi tanımlanmış olmalıdır (örneğin, ABNF kullanılarak veya mevcut parametrelerin sentaksına atıfta bulunarak).
3. **Vendor-Spesifik Parametreler:**
    - Kayıt edilmemiş, vendor-spesifik parametreler için, çakışmaları önlemek adına benzersiz bir önek kullanılmalıdır (ör. `companyname_` ile başlayan adlar).

8.3. Yeni Yetkilendirme Verme Türlerinin Tanımlanması

1. **URI Kullanımı:**
    - Yeni yetkilendirme türleri, **grant_type** parametresi ile benzersiz bir mutlak URI atanarak tanımlanabilir.
2. **Ek Parametreler:**
    - Yeni bir yetkilendirme verme türü, token uç noktasında ek parametreler gerektiriyorsa, bu parametreler OAuth Parameters kayıt defterine kayıt edilmelidir.

8.4. Yeni Yetkilendirme Uç Noktası Yanıt Türlerinin Tanımlanması

1. **Yanıt Türlerinin Kaydı:**
    - Yeni yanıt türleri, **Authorization Endpoint Response Types** kayıt defterine kayıt edilmelidir (Bölüm 11.3).
2. **Yanıt Türü Yapısı:**
    - Yanıt türü adları, **response-type ABNF** kurallarına uymalıdır.
    - Yanıt türü birden fazla değer içeriyorsa (%x20 boşluk karakteri ile ayrılmış), değerlerin sırası önemli değildir. Ancak, yalnızca bir sıra kayıt edilebilir.
3. **Örnek:**
    - "token code" gibi bir yanıt türü tanımlanabilir ve kayıt edilebilir.
    - Ancak aynı kombinasyon farklı bir sıra ile ("code token") tekrar kayıt edilemez.

8.5. Ek Hata Kodlarının Tanımlanması

1. **Yeni Hata Kodları:**
    - Yeni hata kodları, protokol genişletmeleriyle ilgili ek durumlar için tanımlanabilir. Örneğin:
        - Yetkilendirme kodu hata yanıtı.
        - Belirteç hata yanıtı.
        - Kaynağa erişim hata yanıtı.
2. **Kayıt Gereklilikleri:**
    - Eğer hata kodu kayıtlı bir belirteç türü, uç nokta parametresi veya yetkilendirme türü ile ilişkiliyse, kayıt edilmelidir (Bölüm 11.4).
    - Kayıtsız genişletmelerde kullanılan hata kodları ise isteğe bağlı olarak kayıt edilebilir.
3. **Adlandırma Kuralları:**
    - Hata kodları, **error ABNF** kurallarına uygun olmalıdır.
    - Benzersiz bir adlandırma sağlamak için hata kodu genellikle bir önek ile başlamalıdır (ör. `example_invalid`).

Özet

Bu bölüm, OAuth protokolünün genişletilebilirliğini sağlamak için kullanılan prosedürleri tanımlamaktadır. Her bir genişletme türü, kayıt gereklilikleri ve adlandırma standartları ile net bir şekilde yapılandırılmıştır. Bu sayede, genişletmelerin standartlaşması ve uyumluluk kolaylaşır.

9. Native Applications (Yerel Uygulamalar)

Yerel uygulamalar, kaynak sahibinin cihazına yüklenen ve orada çalışan istemcilerdir. Örnek olarak masaüstü uygulamaları ve yerel mobil uygulamalar gösterilebilir. Bu tür uygulamalar, güvenlik, platform yetenekleri ve kullanıcı deneyimi ile ilgili özel dikkat gerektirir. Bölüm, yerel uygulamalar için yetkilendirme uç noktasıyla etkileşim kurma yöntemlerini ve bu süreçte dikkate alınması gereken önemli noktaları ele almaktadır.

Yetkilendirme Uç Noktası ve Kullanıcı-Aracı Etkileşimi

Yerel uygulamalar, kaynak sahibinin kullanıcı-aracısıyla (user-agent, genellikle bir tarayıcı) etkileşim kurarak yetkilendirme uç noktasına erişir. Bu süreç iki ana yöntemle gerçekleşebilir:

1. **Dış Kullanıcı-Aracısı (External User-Agent):**
    - Yerel uygulama, dış bir tarayıcı veya kullanıcı-aracısı kullanır. Bu yöntem, yetkilendirme sunucusunun yanıtını yakalamak için aşağıdaki tekniklerden birini kullanabilir:
        - İşletim sisteminde kayıtlı bir URI şeması kullanarak istemciyi tetiklemek.
        - Kullanıcıdan kimlik bilgilerini kopyalayıp yapıştırmasını istemek.
        - Yerel bir web sunucusu çalıştırmak.
        - Kullanıcı-aracısı uzantısı yüklemek.
        - Sunucunun kontrol ettiği bir yeniden yönlendirme URI’sı sağlayarak yanıtı yerel uygulamaya iletmek.
2. **Gömülü Kullanıcı-Aracısı (Embedded User-Agent):**
    - Yerel uygulama, gömülü bir kullanıcı-aracısı ile doğrudan iletişim kurar ve şu yöntemlerle yanıtı alır:
        - Kaynak yükleme sırasında yayılan durum değişikliklerini izlemek.
        - Kullanıcı-aracısının çerez depolamasına erişmek.

Dış ve Gömülü Kullanıcı-Aracılarının Karşılaştırılması

Yerel uygulama geliştiricileri, dış ve gömülü kullanıcı-aracılarından birini seçerken şu noktaları dikkate almalıdır:

Dış Kullanıcı-Aracısı Avantajları:

- **Tamamlanma Oranı Artışı:**
    - Kullanıcı, yetkilendirme sunucusunda zaten aktif bir oturuma sahipse, yeniden kimlik doğrulama gerekliliği ortadan kalkar.
- **Kullanıcı Deneyimi:**
    - Kullanıcının alışkın olduğu bir arayüz sunar.
- **Ekstra Güvenlik Özellikleri:**
    - Şifre yöneticileri veya iki faktörlü cihaz okuyucuları gibi özellikler kullanıcıya yardımcı olabilir.

Gömülü Kullanıcı-Aracısı Avantajları ve Dezavantajları:

- **Kullanılabilirlik:**
    - Uygulama, bağlam değişikliği olmadan çalışabilir ve yeni pencereler açılması gerekmez.
- **Güvenlik Sorunları:**
    - Kullanıcılar, kimlik doğrulama için kimliği belirsiz bir pencereye güvenmek zorunda kalır. Bu durum, kullanıcıların kimlik doğrulama taleplerine sorgusuz güvenmesini öğreterek phishing (oltalama) saldırılarını kolaylaştırır.

Yetkilendirme Akış Türleri

Yerel uygulamalar, **yetkilendirme kodu** veya **örtük yetkilendirme** (implicit grant) akış türlerinden birini seçebilir. Hangi yöntemin kullanılacağına karar verirken şu noktalar göz önünde bulundurulmalıdır:

1. **Yetkilendirme Kodu Akışı:**
    - Yerel uygulamalar, **istemci kimlik bilgilerini (client credentials)** kullanmadan yetkilendirme kodu akışını tercih etmelidir. Bunun nedeni, yerel uygulamaların istemci kimlik bilgilerini gizli tutamamasıdır.
2. **Örtük Yetkilendirme Akışı:**
    - Örtük akışta, yenileme belirteci (refresh token) döndürülmez. Bu nedenle, erişim belirteci (access token) süresi dolduğunda yetkilendirme işlemi yeniden yapılmalıdır.

Özet

Bu bölüm, yerel uygulamaların OAuth protokolünü kullanarak yetkilendirme işlemlerini nasıl gerçekleştirdiğini açıklar. Geliştiricilerin güvenlik, kullanılabilirlik ve kullanıcı deneyimi arasında denge kurması gerekir. Dış kullanıcı-aracısı genellikle güvenlik ve kullanıcı memnuniyeti açısından daha avantajlıdır, ancak bazı durumlarda gömülü kullanıcı-aracısı kullanılabilir. Ayrıca, yetkilendirme kodu akışı daha güvenli ve esnek bir yöntem sunar, bu yüzden yerel uygulamalarda öncelikli olarak tercih edilmelidir.

10.1. Client Authentication

Yetkilendirme sunucusu, web uygulama istemcileri için istemci kimlik doğrulaması amacıyla istemci kimlik bilgilerini oluşturur. Yetkilendirme sunucusu, istemci parolası gibi daha güçlü istemci kimlik doğrulama yöntemlerini göz önünde bulundurması teşvik edilir. Web uygulama istemcileri, istemci parolalarının ve diğer istemci kimlik bilgilerinin gizliliğini sağlamak ZORUNDADIR. Yetkilendirme sunucusu, istemci kimlik doğrulaması amacıyla yerel uygulama veya kullanıcı-aracısı tabanlı uygulama istemcilerine istemci parolası veya diğer istemci kimlik bilgilerini veremez. Yetkilendirme sunucusu, belirli bir cihazda yerel bir uygulama istemcisinin belirli bir kurulumuna ait bir istemci parolası veya diğer kimlik bilgilerini verebilir.

İstemci kimlik doğrulaması mümkün olmadığında, yetkilendirme sunucusu istemcinin kimliğini doğrulamak için diğer yöntemleri kullanMALIDIR – örneğin, istemci yönlendirme URI'sinin kaydını gerektirmek veya kaynak sahibi ile kimlik doğrulaması yapmak. Kaynak sahibi yetkilendirmesi istendiğinde, geçerli bir yönlendirme URI'si istemcinin kimliğini doğrulamak için yeterli değildir ancak kaynak sahibi yetkilendirmesini aldıktan sonra sahte bir istemciye kimlik bilgilerini teslim etmeyi engellemek için kullanılabilir.

Yetkilendirme sunucusu, kimlik doğrulaması yapılmamış istemcilerle etkileşimde bulunmanın güvenlik etkilerini dikkate almalı ve bu tür istemcilere verilen diğer kimlik bilgilerini (örneğin, yenileme jetonları) potansiyel olarak ifşa olmasını sınırlamak için önlemler almalıdır.

10.2. Client Impersonation

Kötü niyetli bir istemci, başka bir istemciyi taklit edebilir ve taklit edilen istemci, istemci kimlik bilgilerini gizli tutmada başarısız olursa veya bunu yapamayacaksa, korunmuş kaynaklara erişim elde edebilir.

Yetkilendirme sunucusu, mümkün olduğunda istemciyi kimlik doğrulamalıdır. Eğer yetkilendirme sunucusu, istemcinin doğası nedeniyle istemciyi kimlik doğrulayamıyorsa, yetkilendirme sunucusu, yetkilendirme yanıtlarını almak için kullanılan herhangi bir yönlendirme URI'sinin kaydını gerektirmelidir ve böyle potansiyel olarak kötü niyetli istemcilere karşı kaynak sahiplerini korumak için başka yöntemler kullanmalıdır. Örneğin, yetkilendirme sunucusu, kaynak sahibini istemcinin kimliğini ve kökenini belirlemede yardımcı olmak için devreye alabilir.

Yetkilendirme sunucusu, açık bir kaynak sahibi kimlik doğrulaması yapmalı ve kaynak sahibine istemci hakkında bilgi sağlamalı ve talep edilen yetkilendirme kapsamı ve süresi hakkında bilgi sunmalıdır. Kaynak sahibinin, mevcut istemci bağlamında bu bilgileri gözden geçirmesi ve isteği onaylayıp reddetmesi gerekmektedir.

Yetkilendirme sunucusu, istemciyi kimlik doğrulamadan veya tekrar eden isteğin orijinal istemciden gelip gelmediğini doğrulamak için başka önlemler almadıkça, tekrar edilen yetkilendirme isteklerini otomatik olarak işlememelidir (aktif bir kaynak sahibi etkileşimi olmadan).

10.3. Access Tokens

Erişim jetonu kimlik bilgileri (ve herhangi bir gizli erişim jetonu özelliği) iletimi ve depolama sırasında gizli tutulmalı ve yalnızca yetkilendirme sunucusu, erişim jetonunun geçerli olduğu kaynak sunucuları ve erişim jetonunun verildiği istemci arasında paylaşılmalıdır. Erişim jetonu kimlik bilgileri yalnızca TLS kullanılarak iletilmelidir.

İmplicit grant türü kullanıldığında, erişim jetonu URI parçasında iletilir ve bu durum, yetkisiz taraflara erişim jetonunu ifşa edebilir.

Yetkilendirme sunucusu, erişim jetonlarının yetkisiz taraflar tarafından oluşturulmasını, değiştirilmesini veya geçerli erişim jetonları üretmek için tahmin edilmesini engellemelidir.

İstemci, yalnızca gerekli olan minimum kapsamda erişim jetonu talep etmelidir. Yetkilendirme sunucusu, talep edilen kapsamı nasıl yerine getireceğini belirlerken istemci kimliğini dikkate almalıdır ve talep edilenden daha az hakla bir erişim jetonu verebilir.

Bu spesifikasyon, kaynak sunucusunun, kendisine sunulan bir erişim jetonunun, yetkilendirme sunucusu tarafından belirtilen istemciye verilmiş olduğuna dair herhangi bir doğrulama yöntemini sağlamaz.

10.4. Refresh Tokens

Yetkilendirme sunucuları, web uygulaması istemcilerine ve yerel uygulama istemcilerine refresh token (yenileme jetonu) verebilir.

Refresh token'lar iletimde ve depolamada gizli tutulmalı ve yalnızca yetkilendirme sunucusu ile refresh token'larının verildiği istemci arasında paylaşılmalıdır. Yetkilendirme sunucusu, bir refresh token ile istemci arasındaki ilişkiyi korumalıdır. Refresh token'lar yalnızca TLS kullanılarak iletilmelidir.

Yetkilendirme sunucusu, istemci kimliği doğrulaması mümkün olduğunda, refresh token ile istemci kimliği arasındaki ilişkiyi doğrulamalıdır. İstemci kimliği doğrulaması yapılamadığında, yetkilendirme sunucusu refresh token kötüye kullanımını tespit etmek için başka yöntemler kullanmalıdır.

Örneğin, yetkilendirme sunucusu refresh token döndürme (refresh token rotation) mekanizmasını kullanabilir. Bu mekanizmada, her erişim jetonu yenileme yanıtı ile yeni bir refresh token verilir. Önceki refresh token geçersiz kılınır ancak yetkilendirme sunucusu tarafından saklanır. Eğer bir refresh token ele geçirilip, hem saldırgan hem de geçerli istemci tarafından kullanılırsa, biri geçersiz kılınmış refresh token'ı sunacak ve bu durum yetkilendirme sunucusuna güvenlik ihlali hakkında bilgi verecektir.

Yetkilendirme sunucusu, refresh token'ların yetkisiz taraflarca üretilmesini, değiştirilmesini veya geçerli refresh token'lar oluşturmak için tahmin edilmesini engellemelidir.

10.5. Authorization Codes

Yetkilendirme kodlarının iletimi güvenli bir kanal üzerinden yapılmalıdır ve istemci, URI bir ağ kaynağını tanımlıyorsa, yönlendirme URI'siyle TLS kullanımını zorunlu kılmalıdır. Çünkü yetkilendirme kodları, kullanıcı ajanı yönlendirmeleriyle iletildiği için, kullanıcı ajanı geçmişi ve HTTP referans başlıkları aracılığıyla ifşa olabilirler.

Yetkilendirme kodları, kaynak sahibi tarafından yetkilendirme sunucusunda verilen yetkiyi doğrulamak için kullanılan, düz metin taşıyıcı kimlik bilgileri olarak çalışır. Yani, istemci yetkilendirme koduna kaynak sahibi kimlik doğrulaması için güveniyorsa, istemci yönlendirme uç noktası TLS kullanımını zorunlu kılmalıdır.

Yetkilendirme kodları kısa ömürlü ve tek kullanımlık olmalıdır. Yetkilendirme sunucusu, bir yetkilendirme kodunun bir erişim jetonuna dönüştürülmesi için birden fazla deneme gözlemlerse, yetkilendirme sunucusu, tehlikeye atılmış yetkilendirme koduna dayalı olarak daha önce verilen tüm erişim jetonlarını iptal etmeyi denemelidir.

Eğer istemci kimlik doğrulaması yapılabiliyorsa, yetkilendirme sunucuları istemciyi kimlik doğrulamalı ve yetkilendirme kodunun aynı istemciye verilmiş olduğundan emin olmalıdır.

10.6. Authorization Code Redirection URI Manipulation

Yetkilendirme kodu akışı kullanılarak yetkilendirme talep edildiğinde, istemci "redirect_uri" parametresi aracılığıyla bir yönlendirme URI'si belirtebilir. Eğer bir saldırgan, yönlendirme URI'sinin değerini manipüle edebilirse, bu durum, yetkilendirme sunucusunun kaynak sahibi kullanıcı ajanını, yetkilendirme koduyla birlikte saldırganın kontrolündeki bir URI'ye yönlendirmesine neden olabilir.

Bir saldırgan, meşru bir istemcide hesap oluşturabilir ve yetkilendirme akışını başlatabilir. Saldırganın kullanıcı ajanı, erişim izni vermek için yetkilendirme sunucusuna gönderildiğinde, saldırgan meşru istemcinin sağladığı yetkilendirme URI'sını alır ve istemcinin yönlendirme URI'sini, saldırganın kontrolündeki bir URI ile değiştirir. Saldırgan daha sonra kurbanı, manipüle edilmiş bağlantıyı takip etmeye ikna ederek, meşru istemciye erişim izni verir.

Yetkilendirme sunucusuna geldiğinde, kurbana meşru ve güvenilir bir istemci adına normal, geçerli bir istek gösterilir ve kurban bu isteği yetkilendirir. Kurban daha sonra yetkilendirme koduyla birlikte, saldırganın kontrolündeki bir uç noktaya yönlendirilir. Saldırgan, yetkilendirme kodunu istemciye, istemci tarafından sağlanan orijinal yönlendirme URI'sını kullanarak göndererek yetkilendirme akışını tamamlar. İstemci, yetkilendirme kodunu bir erişim jetonuyla değiştirir ve bunu saldırganın istemci hesabına bağlar, bu da artık kurban tarafından yetkilendirilmiş korunan kaynaklara erişim sağlar (istemci aracılığıyla).

Böyle bir saldırıyı önlemek için, yetkilendirme sunucusu, yetkilendirme kodu elde etmek için kullanılan yönlendirme URI'sinin, yetkilendirme kodunu erişim jetonuyla değiştirme sırasında sağlanan yönlendirme URI'siyle tamamen aynı olduğunu doğrulamalıdır. Yetkilendirme sunucusu, kamu istemcilerinin yönlendirme URI'lerini kaydetmelerini zorunlu kılmalı ve özel istemciler için de bunu sağlamak isteğe bağlı olmalıdır. Eğer bir yönlendirme URI'sı talepte sağlanmışsa, yetkilendirme sunucusu bunu kaydedilen değerle karşılaştırarak doğrulamalıdır.

10.7. Resource Owner Password Credentials

Kaynak sahibi şifresi kimlik bilgileri grant tipi, genellikle eski sistemler veya geçiş sebepleriyle kullanılır. Bu, istemcinin kullanıcı adı ve şifreleri depolama riskini azaltır, ancak yine de istemciye yüksek ayrıcalıklı kimlik bilgilerini açıklamak zorunluluğunu ortadan kaldırmaz.

Bu grant tipi, diğer grant tiplerine göre daha yüksek bir risk taşır çünkü bu protokolün kaçınmaya çalıştığı şifre kullanım modelini sürdürmektedir. İstemci, şifreyi kötüye kullanabilir ya da şifre, istemcinin tuttuğu günlük dosyaları veya diğer kayıtlar aracılığıyla istemeden bir saldırgana ifşa olabilir.

Ayrıca, kaynak sahibi yetkilendirme süreci üzerinde kontrol sahibi olmadığı için (kaynak sahibi, kimlik bilgilerini istemciye teslim ettiğinde süreci sonlandırır), istemci, kaynak sahibinin istediğinden daha geniş bir erişim alanı ile erişim jetonları alabilir. Yetkilendirme sunucusu, bu grant tipiyle verilen erişim jetonlarının kapsamını ve ömrünü dikkate almalıdır.

Yetkilendirme sunucusu ve istemci, bu grant tipinin kullanımını en aza indirmeli ve mümkünse diğer grant tiplerini kullanmalıdır.

10.8. Request Confidentiality

Erişim jetonları, yenileme jetonları, kaynak sahibi şifreleri ve istemci kimlik bilgileri açık metin olarak iletilmemelidir. Yetkilendirme kodları açık metin olarak iletilmemelidir.

"State" ve "scope" parametreleri, hassas istemci veya kaynak sahibi bilgilerini açık metin olarak içermemelidir, çünkü bu parametreler güvensiz kanallar üzerinden iletilebilir veya güvensiz bir şekilde depolanabilir.

10.9. Ensuring Endpoint Authenticity

Orta adam (man-in-the-middle) saldırılarını önlemek için, yetkilendirme sunucusu, yetkilendirme ve jeton uç noktalarına gönderilen her istekte tanımlanan sunucu kimlik doğrulamasıyla TLS kullanımını zorunlu kılmalıdır. İstemci, yetkilendirme sunucusunun TLS sertifikasını tanımlandığı şekilde ve sunucu kimliği doğrulama gereksinimlerine uygun olarak doğrulamalıdır.

10.10. Credentials-Guessing Attacks

Yetkilendirme sunucusu, saldırganların erişim token'larını, yetkilendirme kodlarını, yenileme token'larını, kaynak sahibi şifrelerini ve istemci kimlik bilgilerini tahmin etmelerini engellemelidir.

Bir saldırganın üretilen token'ları (ve son kullanıcı tarafından işlenmesi amaçlanmayan diğer kimlik bilgilerini) tahmin etme olasılığı, 2^(-128) veya daha küçük olmalı ve tercihen 2^(-160) veya daha küçük olmalıdır.

Yetkilendirme sunucusu, son kullanıcı kullanımına yönelik kimlik bilgilerini korumak için diğer yöntemleri de kullanmalıdır.

10.11. Phishing Attacks

Bu ve benzer protokollerin yaygın kullanımı, son kullanıcıların şifrelerini girmeleri istenen web sitelerine yönlendirilme pratiğine alışmalarına neden olabilir. Eğer son kullanıcılar, kimlik bilgilerini girmeden önce bu web sitelerinin özgünlüğünü doğrulamakta dikkatli olmazlarsa, saldırganlar bu durumu kaynak sahiplerinin şifrelerini çalmak için kötüye kullanabilirler.

Hizmet sağlayıcıları, son kullanıcıları phishing saldırılarının oluşturduğu riskler hakkında eğitmeye çalışmalı ve son kullanıcıların web sitelerinin özgünlüğünü kolayca doğrulamalarını sağlayacak mekanizmalar sunmalıdır. İstemci geliştiricileri, kullanıcı aracıyla (örneğin, dışsal, gömülü) nasıl etkileşimde bulunduklarının ve son kullanıcının yetkilendirme sunucusunun özgünlüğünü doğrulama yeteneğinin güvenlik etkilerini göz önünde bulundurmalıdır.

Phishing saldırılarının riskini azaltmak için, yetkilendirme sunucuları, son kullanıcı etkileşimi için kullanılan her uç noktada TLS kullanımını zorunlu kılmalıdır.

10.12. Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF), bir saldırganın mağdur son kullanıcısının kullanıcı aracını (örneğin, yanıltıcı bir bağlantı, resim veya yönlendirme olarak kullanıcı aracına sağlanan kötü niyetli bir URI) güvenilir bir sunucuya yönlendirmesine neden olduğu bir saldırıdır (genellikle geçerli bir oturum çereziyle sağlanan bir güvenilir bağlantı üzerinden).

Bir CSRF saldırısı, istemcinin yönlendirme URI'sına karşı yapıldığında, saldırganın kendi yetkilendirme kodunu veya erişim belirtecini enjekte etmesine olanak tanır; bu da istemcinin mağdurun korunmuş kaynakları yerine saldırganın korunmuş kaynaklarıyla ilişkilendirilmiş bir erişim belirteci kullanmasına yol açabilir (örneğin, mağdurun banka hesap bilgilerini saldırganın kontrolündeki korunmuş bir kaynağa kaydetmek).

İstemci, yönlendirme URI'sı için CSRF koruması uygulamak zorundadır. Bu genellikle, yönlendirme URI'sı uç noktasına gönderilen her isteğin, isteği kullanıcı aracının doğrulanmış durumu ile bağlayan bir değer içermesini gerektirerek yapılır (örneğin, kullanıcı aracını doğrulamak için kullanılan oturum çerezinin bir karması). İstemci, yetkilendirme isteği yaparken bu değeri yetkilendirme sunucusuna iletmek için "state" istek parametresini kullanmalıdır.

Yetkilendirme, son kullanıcıdan alındıktan sonra, yetkilendirme sunucusu son kullanıcının kullanıcı aracını gerekli bağlama değerini içeren "state" parametresiyle istemciye geri yönlendirir. Bağlama değeri, istemcinin isteğin geçerliliğini doğrulamasını sağlar, çünkü bağlama değeri, kullanıcı aracının doğrulanmış durumu ile eşleşmelidir. CSRF koruması için kullanılan bağlama değeri, tahmin edilemez bir değer içermelidir (Bölüm 10.10'da açıklandığı gibi) ve kullanıcı aracının doğrulanmış durumu (örneğin, oturum çerezi, HTML5 yerel depolama), yalnızca istemci ve kullanıcı aracı tarafından erişilebilen bir konumda tutulmalıdır (yani, aynı kaynak politikasıyla korunmalıdır).

Yetkilendirme sunucusunun yetkilendirme uç noktasına karşı yapılan bir CSRF saldırısı, saldırganın son kullanıcının bilgisi olmadan kötü niyetli bir istemci için son kullanıcı yetkilendirmesi almasına yol açabilir.

Yetkilendirme sunucusu, yetkilendirme uç noktası için CSRF koruması uygulamak zorundadır ve bir kötü niyetli istemcinin, kaynak sahibinin farkındalığı ve açık onayı olmadan yetkilendirme almasını engellemelidir.

10.13. Clickjacking

Bir clickjacking saldırısında, bir saldırgan geçerli bir istemci kaydeder ve ardından kötü niyetli bir site oluşturur. Bu sitede, yetkilendirme sunucusunun yetkilendirme uç noktası web sayfası, şeffaf bir iframe içinde, dikkatlice yerleştirilmiş sahte düğmelerin üzerine yüklenir. Bu düğmeler, yetkilendirme sayfasındaki önemli düğmelerin hemen altına yerleştirilecek şekilde tasarlanır. Bir son kullanıcı yanıltıcı bir görünür düğmeye tıkladığında, aslında yetkilendirme sayfasındaki görünmeyen bir düğmeye (örneğin "Yetkilendir" düğmesi) tıklamış olur. Bu, saldırganın bir kaynak sahibini, son kullanıcının bilgisi olmadan istemcisine erişim izni vermesini sağlamak için kullanabileceği bir yöntemdir.

Bu tür bir saldırıyı engellemek için, yerel uygulamalar son kullanıcının yetkilendirmesini isterken uygulama içinde tarayıcıları yerleştirmek yerine harici tarayıcılar kullanmalıdır. Çoğu yeni tarayıcıda, iframe'lerin önlenmesi, yetkilendirme sunucusu tarafından (standart olmayan) "x-frame-options" başlığı kullanılarak sağlanabilir. Bu başlığın iki değeri vardır: "deny" ve "sameorigin". "Deny" herhangi bir çerçevelemeyi engellerken, "sameorigin" farklı bir kaynağa sahip siteler tarafından çerçevelemeyi engeller. Eski tarayıcılar için, JavaScript ile yapılan frame-busting (çerçeve kırma) teknikleri kullanılabilir, ancak bu tüm tarayıcılarda etkili olmayabilir.

10.14. Code Injection and Input Validation

Bir kod enjeksiyonu saldırısı, bir uygulamanın dışarıdan gelen bir girdi veya değişkeni temizlemeden kullanarak uygulama mantığını değiştirmesine neden olan bir saldırıdır. Bu, saldırganın uygulama cihazına veya verilerine erişim kazanmasına, hizmet reddi (DoS) saldırısı yapmasına veya geniş bir kötü amaçlı yan etki yelpazesi oluşturmasına olanak tanıyabilir.

Yetkilendirme sunucusu ve istemcisi, alınan her değeri temizlemeli (ve mümkünse doğrulamalıdır) — özellikle "state" ve "redirect_uri" parametrelerinin değerlerini.

10.15. Open Redirectors

Yetkilendirme sunucusu, yetkilendirme uç noktası ve istemci yönlendirme uç noktası yanlış yapılandırılabilir ve açık yönlendirme yapan birer yönlendirici olarak çalışabilirler. Açık

yönlendirici, bir parametre kullanarak kullanıcı aracını, parametre değerinin belirttiği konuma otomatik olarak yönlendiren ve herhangi bir doğrulama yapmayan bir uç noktadır.

Açık yönlendiriciler, kimlik avı saldırılarında veya bir saldırgan tarafından son kullanıcıları kötü amaçlı sitelere yönlendirmek için kullanılabilir; bunun için tanıdık ve güvenilen bir hedefin URI yetki bileşeni kullanılır. Ayrıca, eğer yetkilendirme sunucusu istemcinin yalnızca yönlendirme URI'sının bir kısmını kaydetmesine izin veriyorsa, bir saldırgan istemcinin kontrol ettiği açık yönlendiriciyi kullanarak, yetkilendirme sunucusu doğrulamasını geçecek ancak yetkilendirme kodunu veya erişim token'ını saldırganın kontrolündeki bir uç noktaya gönderecek bir yönlendirme URI'si oluşturabilir.

10.16. Misuse of Access Token to Impersonate Resource Owner in Implicit Flow

Açık akışları kullanan halk istemcileri için bu özellik, bir erişim token'ının hangi istemciye verildiğini belirlemek için herhangi bir yöntem sunmaz.
Bir kaynak sahibi, bir saldırganın kötü niyetli istemcisine erişim token'ı vererek bir kaynağa erişimi isteyerek gönüllü olarak kaynak erişimini devredebilir. Bu, kimlik avı veya başka bir mazeret nedeniyle olabilir. Ayrıca bir saldırgan, başka bir mekanizma aracılığıyla bir token çalabilir. Saldırgan daha sonra, kaynak sahibini taklit etmek için erişim token'ını meşru bir halk istemcisine sunmayı deneyebilir.

Açık akışta (response_type=token), saldırgan, yetkilendirme sunucusundan gelen yanıttaki token'ı kolayca değiştirebilir, gerçek erişim token'ını daha önce saldırgana verilen token ile değiştirebilir.

Arka kanal üzerinden bir erişim token'ı alarak istemcinin kullanıcısını tanımlayan yerel uygulamalarla iletişim kuran sunucular, saldırganın, aralarında token çalınmış bir uygulama yaratarak rastgele çalınmış erişim token'larını enjekte etmesiyle benzer şekilde tehlikeye girebilir.

Kaynağın yalnızca kaynak sahibinin geçerli bir erişim token'ı sunabileceği varsayımında bulunan herhangi bir halk istemcisi, bu tür bir saldırıya karşı savunmasızdır.

Bu tür bir saldırı, kaynak sahibine ait bilgileri meşru istemcide saldırgana (kötü niyetli istemci) açığa çıkarabilir. Ayrıca, saldırganın, erişim token'ını veya yetkilendirme kodunu ilk veren kaynak sahibinin izinleriyle aynı izinlere sahip olarak, meşru istemcide işlemler gerçekleştirmesine olanak tanır.

Kaynak sahiplerini istemcilere kimlik doğrulama süreci, bu özellik kapsamında değildir. Yetkilendirme sürecini istemciler için delege edilmiş bir son kullanıcı kimlik doğrulama biçimi olarak kullanan herhangi bir özellik (örneğin, üçüncü taraf oturum açma hizmeti) ek güvenlik mekanizmaları olmadan açık akışı kullanmamalıdır. Bu mekanizmalar, istemcinin erişim token'ının kendisi için verilip verilmediğini belirlemesini sağlamalıdır (örneğin, erişim token'ını izleyerek).