1. Introduction

RFC 6750'nin **Introduction (Giriş)** bölümü, OAuth 2.0 ile ilgili önemli bir konsepti olan **Bearer Token** kullanımını açıklamaktadır. Bu bölümdeki temel noktaları şu şekilde özetleyebiliriz:

1. **OAuth ve Erişim Token'ları**: OAuth, istemcilerin korunan kaynaklara erişimini sağlamak için **erişim token'ları** almasına olanak tanır. Erişim token'ı, bir kaynağa erişim yetkisi verilen bir **dizedir**. Bu, doğrudan kaynak sahibinin kimlik bilgilerini kullanmak yerine istemcinin kimlik doğrulamasını sağlayan bir yapıdır. RFC 6749'da bu token'lar, istemcinin **erişim yetkisini** temsil eden bir string olarak tanımlanmıştır.
2. **Token Verme Süreci**: Token'lar, bir **yetkilendirme sunucusu** tarafından **kaynak sahibinin onayı** ile istemcilere verilir. İstemci bu erişim token'ını, **kaynak sunucusunda** barındırılan korunan kaynaklara erişim sağlamak için kullanır.
3. **Bearer Token Kullanımı**: Bu belge, OAuth erişim token'larının **Bearer Token** şeklinde kullanımını tanımlar. **Bearer Token**, kimlik doğrulaması yapmak için kullanılan ve taşındığı kişi veya sistem tarafından kolayca kullanılabilen bir token türüdür. Yani, token'ı taşıyan kişi, bu token ile yetkilendirilmiş sayılır.
4. **TLS ve Güvenlik**: **TLS (Transport Layer Security)**, Bearer token'larının güvenli bir şekilde taşınması için **zorunlu** olarak kullanılması gereken bir güvenlik protokolüdür. Bu RFC, HTTP üzerinden Bearer token kullanarak korunan kaynaklara erişmek için TLS kullanımını zorunlu kılar. Diğer protokollerle kullanılmak üzere bu spesifikasyon genişletilebilir, ancak HTTPS (TLS kullanımı) burada ana gerekliliktir.
5. **Bearer Authentication Şeması**: Bu RFC, **Bearer authentication** şemasını tanımlar. Bu şema, özellikle **server authentication (sunucu kimlik doğrulaması)** için **WWW-Authenticate** ve **Authorization** HTTP başlıkları aracılığıyla kullanılmaktadır. Ancak, bu şemanın yalnızca sunucu kimlik doğrulaması için değil, aynı zamanda **proxy kimlik doğrulaması** için de kullanılabileceği belirtilmektedir.

Özetle, RFC 6750, OAuth 2.0 ile verilen Bearer Token'larının HTTP üzerinden, özellikle TLS protokolü ile güvenli bir şekilde taşınarak kullanılması gerektiğini belirtmektedir. Bearer Token'lar, OAuth 2.0'ın bir parçası olarak tasarlanmış olsa da, aslında genel bir HTTP yetkilendirme yöntemi olarak her türlü token için kullanılabilir.

1.1. Notational Conventions

RFC 6750'deki **1.1 Notational Conventions (Notasyon Konvansiyonları)** bölümü, bu dokümanda kullanılan terminoloji ve notasyon kurallarını açıklamaktadır. Bu bölümde belirtilenler, dokümanda kullanılan bazı temel kelimelerin anlamını ve kullanılan notasyon sistemlerini açıklar. İşte önemli noktalar:

1. Anahtar Kelimeler ve Yorumları (RFC 2119):

Bu dokümanda yer alan bazı anahtar kelimeler, **RFC 2119**'da tanımlanan anlamlarına göre kullanılacaktır. Bu anahtar kelimeler, belirli gerekliliklerin ne kadar güçlü olduğunu belirtir. Bu anahtar kelimelerin anlamları şunlardır:

- **MUST / REQUIRED / SHALL**: Zorunlu, mutlaka yapılması gereken bir şey. Bu kelimeler, bir davranışın **zorunlu** olduğunu belirtir.
- **MUST NOT / SHALL NOT**: Yasak, yapılmaması gereken bir şey.
- **SHOULD**: Önerilen, genellikle yapılması gereken ama bazı durumlarda esneklik tanınan bir şey.
- **SHOULD NOT**: Genellikle yapılmaması gereken, ama bazen istisnai durumlar olabilir.
- **RECOMMENDED**: Tavsiye edilen, ama zorunlu olmayan bir şey.
- **MAY / OPTIONAL**: İstediğe bağlı, yapılmasında herhangi bir zorunluluk olmayan bir şey.

Bu kelimelerin her biri, **RFC 2119**'daki tanımlara dayanarak belirli bir gerekliliği ifade eder ve bu gereklilikler, dokümanda belirtilen kuralları netleştirmek için kullanılır.

2. ABNF Notasyonu (RFC 5234):

Bu doküman, protokol tanımlarını ve yapıları ifade etmek için **Augmented Backus-Naur Form (ABNF)** notasyonunu kullanır. **ABNF** notasyonu, metin tabanlı protokollerin yapısal dil tanımlarını belirtmek için kullanılan bir biçimdir. Bu, protokolün nasıl oluşturulacağına dair belirli kuralları tanımlar ve metinler arasındaki geçişleri netleştirir.

3. HTTP/1.1 ve URI Notasyonları:

- **auth-param** ve **auth-scheme**: HTTP/1.1 [RFC2617] belgesinde tanımlanan iki terimdir. Bu terimler, HTTP kimlik doğrulama parametreleri ve şemalarını belirtir. Bu parametreler, kimlik doğrulama isteklerinde kullanılan bilgilerdir.
- **URI-reference**: **Uniform Resource Identifier (URI)** tanımlarına dair kuralları belirleyen **RFC 3986**'dan alınmıştır. URI, kaynakların tanımlanması için kullanılan bir standarttır ve bu terim, URI'nin nasıl yazılacağına dair kuralları belirtir.

4. Büyük/Küçük Harf Duyarlılığı:

Bu belgede, protokol parametrelerinin **isimleri ve değerleri** genellikle **büyük/küçük harfe duyarlıdır**. Yani "token" ile "Token" veya "Redirect_URI" ile "redirect_uri" gibi farklı yazımlar farklı anlamlar taşıyabilir. Bu, protokolün doğru şekilde çalışabilmesi için önemlidir; yanlış yazım hataları protokolün hatalı çalışmasına neden olabilir.

Özet:

Bu bölümde, dokümanda kullanılan terimler ve kurallar netleştirilmektedir. Anahtar kelimeler, yapılması gerekenler ve yapılmaması gerekenler hakkında kesinlik sağlar. Ayrıca, doküman metinlerinin nasıl yorumlanacağı ve protokol parametrelerinin yazım kuralları belirtilmiştir. ABNF notasyonu ve URI notasyonları, protokolün yapılandırılmasında kullanılan teknik detaylardır.

1.2. Terminology

1.2. Terminology (Terminoloji) bölümü, RFC 6750'de kullanılan bazı temel terimlerin tanımlarını sağlar. Burada, özellikle **"Bearer Token"** (taşıyıcı jeton) terimi açıklanmıştır. İşte detaylı açıklama:

Bearer Token (Taşıyıcı Jeton):

- **Bearer Token**, bir güvenlik jetonu türüdür. Bu tür bir jetonun temel özelliği, jetonu elinde bulunduran (yani jetonu taşıyan) herhangi bir kişinin, jetonu kullanma hakkına sahip olmasıdır. Yani, jetonun fiziksel olarak elinde bulunduran kişi, başka birinin jetonu taşımasına benzer şekilde, jetonu kullanabilir.
- **Bearer Token** kullanmak, jetonun sahibinin **kriptoğrafik anahtar materyali** (proof-of-possession) ile jetonun gerçek sahibi olduğunu ispat etmesine gerek olmadığı anlamına gelir. Yani, jetonu elinde bulunduran kişi, jetonu kullanırken özel bir anahtar veya doğrulama gerektirmeden işlem yapabilir.
- **Özetle**, "Bearer Token", sadece jetonu taşıyan kişinin bu jetonu herhangi bir sınırlama olmadan kullanabileceği, basit ve yaygın olarak kullanılan bir güvenlik mekanizmasıdır.

Diğer Terimler:

- RFC 6750'de geçen **diğer terimler**, **"The OAuth 2.0 Authorization Framework"** [RFC6749] belgesinde tanımlandığı şekliyle kullanılır. Yani, OAuth 2.0 çerçevesinde kullanılan ve burada yer almayan diğer terimler için, **RFC 6749**'daki tanımlar geçerli olacaktır.

Özet:

Bu bölümde, "Bearer Token" terimi detaylı bir şekilde açıklanmış ve başka terimler için **RFC 6749**'da belirtilen tanımların geçerli olduğu vurgulanmıştır. Bearer token, herhangi bir doğrulama gerektirmeden sahiplik kanıtı sunmaksızın işlem yapılmasına imkan tanır, bu da OAuth gibi protokollerle güvenli veri erişimi için yaygın bir kullanım sağlar.

1.3. Overview

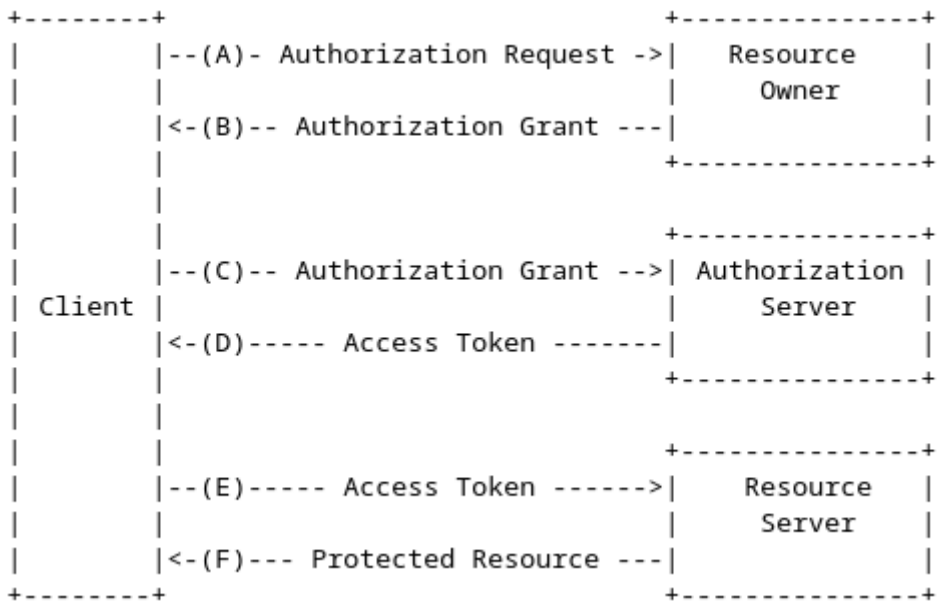
1.3. Overview (Genel Bakış) bölümü, OAuth 2.0 protokolünün genel işleyişini açıklamaktadır. OAuth, istemcilerin (client) korunan bir kaynağa erişebilmesi için bir kaynak sahibi (resource owner) adına hareket etmelerini sağlayan bir mekanizma sunar. Aşağıda, bu bölümün açıklaması yapılmıştır:

OAuth 2.0 Genel Akışı:

OAuth 2.0, istemcilerin bir **kaynak sahibinin** adına **korunan bir kaynağa** erişmesini sağlayan bir yetkilendirme protokolüdür. Ancak, bir istemci bir kaynağa erişmeden önce şu adımları takip etmelidir:

1. **Yetkilendirme Talebi (Authorization Request)**: İstemci, kaynak sahibinden bir yetkilendirme talebi gönderir. Bu talep, genellikle kaynak sahibinin onayını gerektirir.

2. **Yetkilendirme Verisi (Authorization Grant)**: Kaynak sahibi, istemcinin talebini onayladığında bir **yetkilendirme verisi (Authorization Grant)** gönderir. Bu veriler, istemcinin kaynağa erişim için gerekli izni temsil eder.
3. **Erişim Jetonu (Access Token)**: İstemci, yetkilendirme verisini bir **erişim jetonuna (Access Token)** dönüştürmek için **Yetkilendirme Sunucusu'na** başvurur. Erişim jetonu, istemcinin belirli bir süre boyunca kaynağa erişebilmesini sağlayacak yetkileri içerir.
4. **Kaynak Sunucusu ile Erişim (Accessing Resource Server)**: İstemci, **erişim jetonunu** kullanarak, **Kaynak Sunucusu'ndan** korunan kaynakları talep eder.
5. **Kaynak Sunucusu Yanıtı**: Kaynak sunucusu, erişim jetonunu doğrular ve geçerliyse, istemciye istenen korunan kaynağı sunar.



Akış Diyagramı

Yukarıda, OAuth 2.0 işlemi ve ilgili taraflar arasındaki etkileşimi açıklayan bir diyagram yer alır. Diyagramda, istemci (client), kaynak sahibi (resource owner), yetkilendirme sunucusu (authorization server) ve kaynak sunucusu (resource server) arasındaki etkileşimler gösterilmiştir.

Diyagramdaki adımlar şu şekildedir:

- **A (Yetkilendirme Talebi)**: İstemci, kaynak sahibinden bir yetkilendirme talebi gönderir.
- **B (Yetkilendirme Verisi)**: Kaynak sahibi, istemciye bir yetkilendirme verisi gönderir.
- **C (Yetkilendirme Verisi ile Başvuru)**: İstemci, yetkilendirme verisini yetkilendirme sunucusuna ileterek erişim jetonu alır.
- **D (Erişim Jetonu)**: Yetkilendirme sunucusu, istemciye geçerli bir erişim jetonu gönderir.
- **E (Erişim Jetonu ile Kaynağa Erişim)**: İstemci, erişim jetonunu kaynak sunucusuna göndererek korunan kaynağa erişmeye çalışır.
- **F (Korunan Kaynağa Erişim Yanıtı)**: Kaynak sunucusu, erişim jetonunun geçerliliğini kontrol eder ve geçerliyse istemciye korunan kaynağı sunar.

Erişim Jetonu (Access Token) ve Kaynak Sunucusu:

- **Adım E**: İstemci, erişim jetonunu kullanarak kaynak sunucusundan korunan kaynağa erişmeye çalışır. Bu aşamada, erişim jetonu kimlik doğrulama amacıyla sunucuya gönderilir.
- **Adım F**: Kaynak sunucusu, erişim jetonunu doğrular. Eğer jeton geçerliyse, istemciye istenilen kaynak sunulur.

Semantik Gereksinimler:

Bu belgede, **adım D** (erişim jetonunun elde edilmesi) ile ilgili olarak, döndürülen erişim jetonunun semantik gereksinimleri belirlenmiştir. Bu, erişim jetonunun doğru ve güvenli bir şekilde kullanılmasını sağlamak için gerekli olan kuralları tanımlar.

Özet:

Bu bölüm, OAuth 2.0'ın genel işleyişini tanımlar ve erişim jetonunun kullanımını özetler. Erişim jetonu, OAuth protokolünde kaynaklara erişimi temsil eden bir anahtar görevi görür. Ayrıca, erişim jetonunun geçerliliğinin doğrulanması ve güvenli bir şekilde kaynak sunucusu ile etkileşim sağlanması gerektiği vurgulanır.

2. Authenticated Requests

Bu bölüm, kaynak sunucularına yapılan kaynak taleplerinde bearer (taşıyıcı) erişim token'larının gönderilmesinin üç yolunu tanımlar. İstemciler, her istekte token'ı iletmek için birden fazla yöntem kullanmamalıdır.

2.1. Authorization Request Header Field

Bu bölüm, **"Authorization" başlık alanı** kullanılarak bir *Bearer* (taşıyıcı) erişim token'ının nasıl gönderileceğini açıklar.

Temel Konsept

İstemci, HTTP/1.1 [RFC2617] tarafından tanımlanan **"Authorization" başlık alanını** kullanarak erişim token'ını iletir. Bu işlem sırasında **"Bearer" kimlik doğrulama şeması** kullanılır. Yani, istemci bir HTTP isteği gönderirken bu başlık alanına *Bearer token* ekler.

Örnek Kullanım

Aşağıda bir GET isteği örneği verilmiştir:

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

Bu örnekte:

- `Authorization: Bearer mF_9.B5f-4.1JqM` kısmı, taşıyıcı token'ın başlıkta nasıl iletildiğini gösterir.
- `mF_9.B5f-4.1JqM` bir erişim token'ıdır.

Söz Dizimi

Bearer kimlik bilgileri için aşağıdaki söz dizimi kullanılır:

- **`b64token`**:
  - Bu, bir Base64 kodlu dizidir ve aşağıdaki karakterleri içerir:
    - Harfler: `A-Z`, `a-z`
    - Rakamlar: `0-9`
    - Özel karakterler: `-`, `.`, `_`, `~`, `+`, `/`
    - Eşittir işareti (`=`) ise padding için kullanılır.
- **`credentials`**:
  - "Bearer" kelimesinden sonra bir veya daha fazla boşluk (`SP`) gelir ve ardından `b64token` eklenir.

Örnek:

```
Bearer mF_9.B5f-4.1JqM
```

Teknik Detaylar ve Gereklilikler

1. **Uyumluluk**:
    - Söz dizimi [RFC2617] ile uyumludur, ancak tam olarak genel HTTP kimlik doğrulama çerçevesiyle örtüşmeyebilir. Bu, mevcut sistemlerin kullanımına uygun olacak şekilde tasarlanmıştır.
2. **Öneriler**:
    - **İstemciler**: Kimlik doğrulamalı istekler yaparken Bearer token'larını **"Authorization" başlık alanında** kullanmalıdır.
    - **Kaynak Sunucuları**: Bu yöntemi desteklemek zorundadır (*MUST* ifadesi ile zorunlu tutulmuştur).

Neden Önemli?

- Bu yöntem, taşıyıcı token'ların güvenli ve standart bir şekilde iletilmesini sağlar.
- Token gönderimi başlık alanında yapıldığı için URI'lerde (örneğin, query string'lerde) token sızması riski azaltılır.

Bu şekilde, Bearer token kullanımı ile kimlik doğrulama işlemleri daha güvenli ve yapılandırılmış hale gelir.

2.2. Form-Encoded Body Parameter

Bu bölümde, erişim token'larının HTTP isteğinin gövdesinde nasıl gönderileceği açıklanır. Token'lar **"access_token"** parametresi olarak kodlanır ve HTTP isteğinin gövdesine eklenir. Ancak bu yöntem belirli şartlara uygun olarak kullanılmalıdır.

Kullanım Koşulları

Erişim token'ını HTTP gövdesinde iletmek için aşağıdaki koşulların hepsi karşılanmalıdır:

1. **`Content-Type` Başlık Alanı**:
    - İstek başlığında `Content-Type: application/x-www-form-urlencoded` olmalıdır. Bu, form kodlama formatını belirtir.
2. **Kodlama Gereklilikleri**:
    - Gövde, HTML 4.01'in application/x-www-form-urlencoded içerik türü için belirlediği kodlama kurallarına uymalıdır.
3. **Tek Parça Gövde**:
    - HTTP isteği tek parçalı bir gövdeye sahip olmalıdır. Yani birden fazla form-data ya da karmaşık gövde yapıları olmamalıdır.
4. **ASCII Karakterleri**:
    - Gövdede kodlanacak içerik tamamen ASCII karakterlerinden oluşmalıdır.

5. **Geçerli HTTP Yöntemi**:
    - Gövde içeriği anlamlı olan bir HTTP yöntemi kullanılmalıdır. Özellikle `GET` yöntemi bu amaçla kullanılamaz. Örneğin, `POST`, `PUT`, veya `PATCH` kullanılabilir.

Parametrelerin Ayrımı

- Gövde, `access_token` parametresiyle birlikte başka istek parametreleri de içerebilir. Bu durumda, tüm parametreler birbirinden doğru şekilde `&` karakteriyle ayrılmalıdır.

Örnek İstek:

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

access_token=mF_9.B5f-4.1JqM
```

Burada:

- `access_token=mF_9.B5f-4.1JqM` kısmı, token'ın nasıl gövdeye eklendiğini gösterir.

Öneriler ve Gereklilikler

1. **Kullanım Önerisi**:
    - Bu yöntem, `Authorization` başlık alanına erişim imkanı olmayan tarayıcılar veya istemciler için tercih edilmelidir.
2. **Kaynak Sunucuları**:
    - Bu yöntemi desteklemek zorunda değildir ancak **destekleyebilir** (MAY). Yani, opsiyoneldir.
3. **Sınırlamalar**:
    - Token'lar gövdede iletildiğinde, başka kodlama yöntemleri ya da başlık tabanlı yetkilendirme kullanımı mümkün olmayabilir. Bu yöntem, yalnızca belirli durumlarda önerilir.

Avantajlar ve Dezavantajlar

- **Avantajlar**:
    - Tarayıcıların `Authorization` başlık alanına erişim kısıtlı olduğunda işe yarar.
    - Gövde üzerinden token iletimi daha sade bir yöntem sunar.
- **Dezavantajlar**:
    - Başlık üzerinden token gönderimine göre güvenlik riskleri daha yüksektir.
    - Token'ın, gövde dışında diğer parametrelerle doğru ayrılmasını sağlamak gereklidir.

Bu yöntem, `application/x-www-form-urlencoded` içeriğini destekleyen uygulama bağlamlarında kullanılabilir. Ancak, güvenlik ve kolaylık açısından, genellikle **"Authorization" başlık alanı** yöntemi tercih edilir.

2.3. URI Query Parameter

Bu bölüm, erişim token'larının URI sorgu parametresi olarak gönderilmesini açıklar. Token, **`access_token`** parametresiyle URI'ye eklenir ve istek bu şekilde yapılır. Ancak bu yöntem, güvenlik zafiyetleri nedeniyle **önerilmez** ve yalnızca başka yollar mümkün olmadığında kullanılmalıdır.

Kullanım Detayları

1. **Erişim Token'ını URI'ye Eklemek**:
    - `access_token` parametresi URI'nin sorgu bileşenine eklenir. Bu yöntemle erişim token'ı bir HTTP GET isteğinde URI'ye dahil edilebilir.

Örnek İstek:

```
GET /resource?access_token=mF_9.B5f-4.1JqM HTTP/1.1
Host: server.example.com
```

Bu istek, erişim token'ını `access_token` parametresi olarak URI içinde taşır.

2. **Diğer Parametrelerle Kullanım**:
    - URI başka parametreler içeriyorsa, `access_token` parametresi diğer parametrelerden **&** karakteriyle ayrılır.

Örnek:

```
https://server.example.com/resource?access_token=mF_9.B5f-4.1JqM&p=q
```

Bu örnekte:

- `access_token`: Token'ı taşır.
- `p=q`: URI'ye ait başka bir sorgu parametresidir.

Öneriler ve Gereklilikler

1. **Önbellek Kontrolü**:
    - Token'ın URI'de taşınması güvenlik zafiyetlerine neden olabileceğinden, istemciler HTTP başlıklarında **`Cache-Control: no-store`** ayarını göndermelidir.
    - Kaynak sunucuları, başarılı (2XX) yanıtlarında **`Cache-Control: private`** başlığı göndermelidir. Bu, token'ın önbelleğe alınmasını engellemeye veya sınırlandırmaya yardımcı olur.
2. **Kullanım Durumu**:

- URI yöntemi, yalnızca token'ı `Authorization` başlık alanı veya HTTP gövdesi içinde taşımak mümkün olmadığında kullanılmalıdır.

3. **Kaynak Sunucuları**:
    - Kaynak sunucuları bu yöntemi desteklemek zorunda değildir ancak **destekleyebilir** (MAY).

Güvenlik Zafiyetleri

1. **URL Günlükleme**:
    - URI içinde taşınan token'lar genellikle web sunucuları, proxy'ler ve tarayıcılar tarafından günlüklenir. Bu durum token'ın kötüye kullanılma riskini artırır.
2. **URI Ad Alanı**:
    - `access_token` parametresi, URI'de bir sorgu parametresi olarak taşınır. Ancak bu, URI ad alanı uygulamalarına aykırıdır ve kötü bir uygulama olarak kabul edilir.
3. **Tavsiyeler**:
    - Bu yöntem, güvenlik zafiyetleri nedeniyle yalnızca **dökümantasyon amaçlı** dahil edilmiştir ve **kullanımı önerilmez**.

Avantajlar ve Dezavantajlar

Avantajlar:

- Basit uygulama: Erişim token'ı URI'ye kolayca eklenebilir.
- Sunucular arasında taşınması daha az karmaşıktır.

Dezavantajlar:

- **Güvenlik riski**: Token'ın günlüklere kaydedilme olasılığı yüksektir.
- URI ad alanı uygulamalarıyla çelişir.
- Önbelleğe alınma riski vardır.

Sonuç olarak, URI yöntemi kullanım kolaylığı sağlasa da, güvenlik riskleri nedeniyle **son çare** olarak düşünülmelidir. Token'lar için genellikle `Authorization` başlık alanı veya HTTP gövdesi tercih edilmelidir.

3. The WWW-Authenticate Response Header Field

Bu bölüm, **`WWW-Authenticate`** HTTP yanıt başlığının nasıl kullanılacağını ve ne gibi bilgileri içermesi gerektiğini açıklar. Bu başlık, istemciye korunan bir kaynağa erişemediğini ve neden erişemediğini bildirir.

Kullanım Detayları

1. **Genel Amaç**:
    - Bir istemcinin, korunan bir kaynağa erişmek için uygun bir kimlik doğrulama bilgisi sunmadığı veya erişim token'ının geçerli olmadığı durumlarda, sunucu **`WWW-Authenticate`** başlığını yanıtında eklemelidir.
    - Bu başlık, istemciye neden başarısız olduğunu ve ne tür bir kimlik doğrulama gerektiğini belirtir.

Yapı ve İçerik

1. **Auth-Scheme (Kimlik Doğrulama Şeması)**:
    - Bu şema her zaman `"Bearer"` olmalıdır.
    - "Bearer" şeması, istemcinin bir erişim token'ı sağlayarak kimliğini doğrulaması gerektiğini belirtir.
2. **Auth-Param (Kimlik Doğrulama Parametreleri)**:
    - **`realm`**:
        - Kaynağın koruma kapsamını tanımlamak için kullanılır. Bu isteğe bağlıdır ve bir başlıkta yalnızca bir kez bulunabilir.
        - Örnek: `realm="example"`
    - **`scope`**:
        - Erişim token'ının hangi kapsamda geçerli olduğunu tanımlar. Kapsamlar, boşlukla ayrılmış değerlerden oluşur.
        - Örnek: `scope="openid profile email"`
        - Bu parametre programatik kullanım için tasarlanmıştır ve son kullanıcılara gösterilmemelidir.
    - **`error`**:
        - İstemciye, isteğin neden reddedildiğini belirtir. Örneğin, token'ın geçersiz olması gibi.
        - Örnek: `error="invalid_token"`
    - **`error_description`**:
        - İstemci geliştiricileri için insan tarafından okunabilir bir açıklama sağlar. Bu, son kullanıcıya gösterilmez.
        - Örnek: `error_description="The access token expired"`

- **`error_uri`**:
    - Hata hakkında daha fazla bilgi sağlayan bir web sayfasının URL'sini belirtir.
    - Örnek: `error_uri="https://example.com/error-info"`

Kullanım Örnekleri

1. **Kimlik Doğrulama Bilgisi Yok**:
    - İstemci, kimlik doğrulama bilgisi sağlamadan bir kaynak talep ederse, sunucu şu şekilde yanıt verir:

      ```
      HTTP/1.1 401 Unauthorized
      WWW-Authenticate: Bearer realm="example"
      ```

2. **Geçersiz Token**:
    - İstemci, geçerliliğini yitirmiş bir token ile erişim talep ettiğinde:

      ```
      HTTP/1.1 401 Unauthorized
      WWW-Authenticate: Bearer realm="example",
                        error="invalid_token",
                        error_description="The access token expired"
      ```

Teknik Detaylar ve Sınırlamalar

1. **Karakter Sınırlamaları**:
    - **`scope`**, **`error`**, **`error_description`** ve **`error_uri`** parametreleri belirli karakter kümeleriyle sınırlıdır:
        - **`scope`**: ASCII %x21, %x23-5B, %x5D-7E (alfanümerik ve belirli semboller) ve boşluk (%x20).
        - **`error`**, **`error_description`**: ASCII %x20-21, %x23-5B, %x5D-7E.
        - **`error_uri`**: URI-referans sentaksına uygun olmalıdır (%x21, %x23-5B, %x5D-7E).
2. **Zorunluluklar ve Öneriler**:
    - **`realm`** ve **`scope`** kullanımı isteğe bağlıdır.
    - **`error`**, yalnızca bir token ile yapılan başarısız isteklerde kullanılmalıdır.

Güvenlik ve Kullanım

- **Güvenlik**:
    - **`WWW-Authenticate`** başlığı, istemciye yalnızca ihtiyaç duyduğu kadar bilgi vermelidir.

        - Geliştiriciye yönelik açıklamalar (**`error_description`**) son kullanıcılara gösterilmemelidir.
    - **Kullanım Önerisi**:
        - **`realm`** ve **`scope`** gibi parametreler, istemcinin erişim gereksinimlerini anlamasını kolaylaştırır.
        - Hatalı veya reddedilen isteklerde **`error`**, **`error_description`** ve **`error_uri`** parametrelerinin kullanılması önerilir.

Bu mekanizma, istemcinin kimlik doğrulama başarısızlıklarının nedenlerini anlamasını ve buna göre düzeltici önlemler almasını sağlar.

3.1. Error Codes

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
{
  "error": "invalid_request",
  "error_description": "The request is missing the 'access_token' parameter."
}
```

2. invalid_token (Geçersiz Token)

- **Açıklama**: Bu hata, gönderilen erişim token'ının geçersiz olduğunu gösterir. Bunun birkaç nedeni olabilir:
    - Erişim token'ı süresi dolmuş olabilir (expired).
    - Token iptal edilmiş olabilir (revoked).
    - Token hatalı (malformed) veya geçersiz bir biçimde gönderilmiş olabilir.
- **Yanıt Kodu**: HTTP 401 (Unauthorized) kullanılmalıdır. Bu durumda istemci, geçerli bir token almak için yetkilendirme sağlayıcıya yeniden başvurmalı ve isteği tekrar göndermelidir.

Örnek:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                  error="invalid_token",
                  error_description="The access token expired."
```

3. insufficient_scope (Yetersiz Erişim Yetkisi)

- **Açıklama**: Bu hata kodu, erişim token'ının, istenen kaynağa erişim için yeterli yetkilere sahip olmadığını belirtir. Yani, istemciye verilen erişim token'ı, istenen işlemi gerçekleştirebilmek için gerekli olan **scope** (erişim izinleri) düzeyine sahip değildir.
- **Yanıt Kodu**: HTTP 403 (Forbidden) kullanılır. Bu durumda, istemciye istenen kaynağa erişim için gerekli olan izinler veya **scope** belirtilir. İstemci, daha yüksek erişim yetkisi (daha geniş scope) için yeni bir token isteyebilir.

Örnek:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
{
  "error": "insufficient_scope",
  "error_description": "The access token does not have the required scope to
access this resource.",
  "scope": "read write"
}
```

4. Kimlik Doğrulama Bilgisi Eksik Olduğunda

Eğer istemci, kimlik doğrulama bilgilerini göndermezse (örneğin, istemci kimliğinin ne olduğunu bilmeden veya geçersiz bir kimlik doğrulama yöntemi kullanarak), bu durumda **hata kodu** verilmeyebilir. Kaynak sunucu, isteği reddeder ancak hata kodu vermez. Bu durumda 401 Unauthorized yanıtı verilmesi beklenir.

Örnek:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example"
```

Sonuç:

- **invalid_request**: İstek eksik, hatalı veya yanlış parametre içeriyor.
- **invalid_token**: Erişim token'ı geçersiz, süresi dolmuş, iptal edilmiş veya hatalı.
- **insufficient_scope**: Token'ın sahip olduğu yetkiler, istenen kaynağa erişim için yeterli değil.

Her bir hata, istemcinin ne yapması gerektiği hakkında bilgi verir ve doğru HTTP durum koduyla birlikte dönülür. Bu hata yönetimi, güvenlik ve kullanım kolaylığı sağlar, böylece istemciler hataları düzgün bir şekilde ele alabilir.

4. Example Access Token Response

Bu bölüm, bir **access token** (erişim token'ı) yanıtının tipik bir örneğini açıklamaktadır. OAuth 2.0 protokolüne göre, istemciler bir kaynağa erişim talebinde bulunduklarında, erişim token'ı genellikle bu yanıtın bir parçası olarak döndürülür. Bu yanıt, istemcinin belirtilen kaynağa erişim için gereken izinlere sahip olup olmadığını belirler.

Örnek Access Token Yanıtı:

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

{
  "access_token": "mF_9.B5f-4.1JqM",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Yanıtın Açıklaması:

1. HTTP/1.1 200 OK

- Bu, HTTP durumu yanıtıdır ve **isteğin başarılı** olduğunu belirtir.
- **200 OK** durumu, sunucunun isteği doğru şekilde işlediğini ve yanıtı başarıyla döndürdüğünü gösterir.

2. Content-Type: application/json;charset=UTF-8

- Bu başlık, yanıtın içeriğinin **JSON formatında** olduğunu belirtir.
- JSON, bu yanıtın veri formatıdır ve istemcinin veriyi nasıl işleyeceğini belirler.

3. Cache-Control: no-store

- Bu başlık, istemcinin veya sunucunun yanıtı **önbelleğe almasını engeller**.
- Önbellekleme genellikle güvenlik riski oluşturabilir (örneğin, token'lar kötüye kullanılabilir), bu nedenle `no-store` başlığı kullanılarak yanıtın depolanması engellenir.

4. Pragma: no-cache

- Bu da bir HTTP başlığıdır ve eski HTTP sürümleriyle uyumluluk amacıyla eklenir.
- **`no-cache`** ifadesi, yanıtın önbelleğe alınmaması gerektiğini belirtir, bu başlık özellikle HTTP/1.0 istemcileri için önemlidir.

JSON Cevabı:

Yanıt gövdesi JSON formatında döndürülmüştür ve aşağıdaki parametreler içerir:

1. access_token

- **Erişim Token'ı**: Bu, istemcinin korunmuş bir kaynağa erişim sağlamak için kullanacağı **token**'dır.

- Örnekte: `"access_token": "mF_9.B5f-4.1JqM"` şeklinde belirtilmiş. Bu token, belirli bir süre geçerli olacaktır ve doğru kimlik doğrulaması yapılmadığı sürece korunmuş kaynağa erişim sağlanamaz.

2. token_type

- **Token Türü**: Bu parametre, token türünü belirtir. OAuth 2.0 protokolü için bu genellikle **"Bearer"** olarak belirtilir.
- "Bearer" token türü, istemcinin bu token'ı her istekle birlikte göndermesi gerektiğini belirtir. Bu tür token'lar, kullanıcı adı veya şifre gerektirmez, sadece token yeterlidir.

3. expires_in

- **Token'ın Geçerlilik Süresi**: Bu, token'ın geçerlilik süresi ile ilgili bilgidir. Örneğin, `"expires_in": 3600` bu token'ın **3600 saniye** (1 saat) boyunca geçerli olduğu anlamına gelir.
- Bu sürenin sonunda token geçersiz olur ve istemcinin yeni bir token alması gerekir.

4. refresh_token

- **Refresh Token**: Bu, istemcinin yeni bir **access token** almak için kullanabileceği bir **refresh token**'dır.
- Örneğin: `"refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"` şeklinde verilmiştir. **Refresh token**, access token süresi sona erdiğinde istemcinin yeniden kimlik doğrulama yapmadan yeni bir token almasına olanak tanır.

Özet:

Bu yanıt, OAuth 2.0'a göre istemcinin kaynağa erişebilmesi için gereken **access token**'ı, token türünü, geçerlilik süresini ve **refresh token**'ı sağlar. İstemci, token'ı kullanarak kaynaklara erişim talep edebilir ve token süresi dolduğunda refresh token ile yenileyebilir.

5. Security Considerations

Bu bölüm, bearer token'ların kullanımı sırasında token yönetimi ile ilgili güvenlik tehditlerini açıklar ve bu tehditleri nasıl hafifletebileceğimizi anlatır.

5.1. Security Threats

Aşağıdaki liste, bazı token türlerini kullanan protokollerle ilgili yaygın tehditleri sunmaktadır. Bu tehditler, NIST Özel Yayın 800-63 [NIST800-63]'e dayanmaktadır. Bu belge, OAuth 2.0 Yetkilendirme spesifikasyonunu [RFC6749] temel aldığından, burada veya ilgili belgelerde açıklanan tehditler tartışılmamaktadır.

- **Token üretimi/değiştirilmesi**: Bir saldırgan, sahte bir token üretebilir veya mevcut bir token'ın içeriğini (örneğin, kimlik doğrulama veya öznitelik beyanları) değiştirerek, kaynak sunucusunun istemciye uygunsuz erişim vermesini sağlayabilir. Örneğin, bir saldırgan token'ın geçerlilik süresini uzatacak şekilde değiştirebilir; kötü niyetli bir istemci, erişememesi gereken bilgilere ulaşabilmek için beyanı değiştirebilir.
- **Token açıklanması**: Token'lar, hassas bilgileri içeren kimlik doğrulama ve öznitelik beyanları taşıyabilir.
- **Token yönlendirmesi**: Bir saldırgan, bir kaynak sunucusu için oluşturulmuş bir token'ı, yanlışlıkla token'ı kendi için geçerli sanan başka bir kaynak sunucusuna erişim sağlamak amacıyla kullanabilir.
- **Token tekrar kullanımı (replay)**: Bir saldırgan, daha önce bir kaynak sunucusunda kullanılmış bir token'ı tekrar kullanmaya çalışabilir.

Bu tehditler, token'ların güvenliğini tehdit edebilir ve doğru yönetilmediklerinde ciddi güvenlik açıklarına yol açabilir.

5.2. Threat Mitigation

Bu bölüm, **token güvenliğiyle ilgili tehditlerin nasıl azaltılabileceğini** anlatıyor. Token, OAuth 2.0 gibi yetkilendirme protokollerinde kullanıcı doğrulama işlemleri için kullanılan bir tür kimlik doğrulama aracıdır. Bu tür token'lar, güvenliği tehdit eden bir dizi saldırıya maruz kalabilir. Burada bahsedilen güvenlik tehditleri ve bunlara karşı alınacak önlemler şu şekilde özetlenebilir:

1. Token İçeriği Koruma

Token'ların, kötü niyetli kişiler tarafından değiştirilmesini engellemek için **dijital imza** veya **Mesaj Kimlik Doğrulama Kodu (MAC)** gibi yöntemler kullanılması gerekir. Ayrıca, token'ın içeriğini doğrudan şifrelemek yerine, token sadece **yetkilendirme bilgilerine referans** içerebilir. Ancak, bu referansların tahmin edilmesi imkansız olmalıdır. Böylece token'da bulunan hassas bilgiler dışarıya sızmaz.

2. TLS (Transport Layer Security) Uygulaması

Güvenli iletişim sağlamak için **TLS** (yani HTTPS) kullanılması zorunludur. Token’lar genellikle hassas bilgiler içerdiğinden, bu bilgilerin şifrelenmesi gereklidir. Eğer bir istemci, token'ı bir kaynağa gönderiyorsa, bu istemci ile kaynak sunucu arasındaki iletişimde **TLS** kullanılarak veri güvenliği sağlanmalıdır. Bu sayede, arada kalabilecek kötü niyetli kişiler token'ı çalamaz.

3. Token Yönlendirmesi (Token Redirect)

Token, yanlışlıkla başka bir kaynak sunucuya yönlendirilirse, o sunucu token'ı yanlış bir şekilde kabul edebilir. Bu durumu engellemek için, token’lar sadece belirli bir **kaynak sunucuya** yönelik olmalıdır. Bunun için, token’ın içinde **hedef kitle (audience)** bilgisi yer almalıdır. Token, yalnızca belirtilen sunucu tarafından kullanılabilmelidir.

4. Token İptali ve Yeniden Kullanılmasını Engelleme

Token’lar kötü niyetli kişiler tarafından ele geçirilebilir. Bu tür saldırılara karşı, token'ların **ömrü kısa tutulmalıdır**. Örneğin, token sadece bir saat geçerli olabilir. Bu şekilde token’lar çalındığında, saldırganın token’ı kullanma süresi sınırlı olur ve daha az zarar verir.

5. Cookie'ler ve Token'lar

Bearer token'lar **çerezlerde** saklanmamalıdır çünkü çerezler genellikle şifrelenmeden iletilir ve bu da güvenlik riskine yol açar. Çerezler kullanılırsa, içinde token bulunan çerezler şifrelenmeli ve güvenli bir şekilde iletilmelidir.

6. Token’ların Güvenliği

Token'lar, **TLS** ile korunmalı, çalınmalarını önlemek için ek güvenlik önlemleri de uygulanmalıdır. Ayrıca, token’ların doğruluğu ve geçerliliği sürekli kontrol edilmelidir. Bir token’ın geçtiği her noktada, token’ı kabul eden sunucu, token’ın geçerli olduğundan emin olmalıdır.

Özetle, bu bölümde **token'ların güvenliğini sağlamak için** çeşitli teknik önlemlerden bahsedilmektedir. Bu önlemler arasında, token’ların şifrelenmesi, yalnızca belirli bir sunucuya yönlendirilmesi, kısa ömürlü olmaları ve güvenli iletişim protokollerinin kullanılması gibi stratejiler bulunur. Bu tür önlemler, token'ların kötüye kullanımını ve ele geçirilmesini engellemeye yardımcı olur.

5.3. Summary of Recommendations

Bu bölüm, **bearer token** (taşıyıcı token) kullanırken güvenliği sağlamak için alınması gereken bazı temel önlemleri özetlemektedir. Her biri, token'ların güvenliğini korumak ve kötüye kullanımı engellemek amacıyla belirli bir güvenlik tehditiyle mücadele etmek için tasarlanmıştır. Özetle öneriler şu şekildedir:

1. Bearer Token'ları Koruyun

Bearer token'lar, kötü niyetli kişiler tarafından ele geçirilirse, korunması gereken kaynaklara yetkisiz erişim sağlanabilir. Bu nedenle, client (istemci) uygulamaları, token’ların yanlış kişilere

sızmasını engellemek zorundadır. Token’ın güvenliğini sağlamak, tüm güvenlik önerilerinin temelini oluşturur.

2. TLS Sertifika Zincirlerini Doğrulayın

İstemci, **korunan kaynaklara** istek gönderirken, TLS (Transport Layer Security) sertifika zincirini doğrulamak zorundadır. Eğer sertifika doğrulaması yapılmazsa, DNS hijacking (DNS yönlendirme) gibi saldırılarla token çalınabilir ve istemci yetkisiz bir şekilde kaynaklara erişebilir. Bu, özellikle man-in-the-middle (ortada adam) saldırılarına karşı korunma sağlar.

3. Her Zaman TLS (https) Kullanın

Bearer token'larla yapılan isteklerde **her zaman TLS (https)** kullanılmalıdır. TLS, iletişimin güvenliğini sağlar ve token'ı ağ üzerinden sızdırılmaya karşı korur. Eğer TLS kullanılmazsa, token’a yönelik pek çok saldırı türü, kötü niyetli kişilerin korunan kaynaklara erişmesine yol açabilir.

4. Bearer Token'ları Çerezlerde Saklamayın

Bearer token'lar, **çerezlerde** (cookies) şifrelenmeden saklanmamalıdır. Çünkü çerezler, çoğu zaman şifrelenmeden iletilir ve böylece kötüye kullanım riski doğar. Eğer token'lar çerezlerde saklanacaksa, **Cross-Site Request Forgery (CSRF)** saldırılarına karşı korunmaya yönelik önlemler alınmalıdır. Çerezlerde saklanmaması, token’ın güvenliği açısından önemlidir.

5. Kısa Süreli Bearer Token'ları Verin

Token sunucuları, genellikle kısa süreli **bearer token'lar** (1 saat veya daha az geçerli) vermelidir. Özellikle web tarayıcısında veya başka güvenlik riski taşıyan ortamlarda çalışan istemciler için kısa süreli token'lar kullanmak, token’ın sızdırılmasının etkilerini azaltır.

6. Scoping (Kapsam) İçeren Bearer Token'ları Verin

Token sunucuları, token'ların yalnızca belirli bir **hedef kitleye** (audience) yönelik kullanılabilmesi için **scope** (kapsam) içeren token'lar vermelidir. Bu, token'ların yanlış kişilere veya sunuculara verilmesini engeller.

7. Bearer Token'ları Sayfa URL'lerinde Geçirmeyin

Bearer token'lar, **sayfa URL'leri** içinde (örneğin, query string parametreleri olarak) gönderilmemelidir. Bunun yerine, token'lar **HTTP başlıkları** veya mesaj gövdelerinde güvenli bir şekilde iletilmelidir. Çünkü, web tarayıcıları, web sunucuları ve diğer yazılımlar, URL'leri (tarayıcı geçmişi, web sunucu günlükleri gibi) yeterince güvenli tutamayabilir. URL içinde token taşıma, kötü niyetli kişilerin geçmiş verilerden, loglardan veya diğer güvensiz alanlardan token’ı çalmalarına olanak tanır.

Özetle:

Bu öneriler, token'ların güvenliğini sağlamak için önemli pratik ipuçları sunmaktadır. **TLS kullanımı**, **kısa süreli token'lar**, **token'ların URL'de taşınmaması** ve **token'ların çerezlerde saklanmaması** gibi güvenlik önlemleri, saldırılara karşı güçlü bir koruma sağlar. Bu öneriler, token'ların kötüye kullanılmasının önlenmesinde kritik öneme sahiptir.