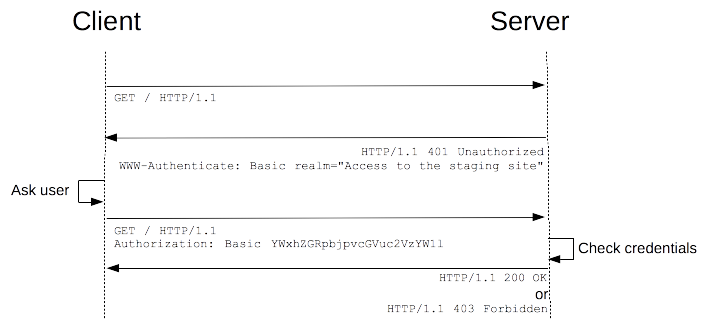
原文地址：https://www.cnblogs.com/RainingNight/p/jwtbearer-authentication-in-asp-net-core.html

在现代Web应用程序中，通常会使用Web, WebApp, NativeApp等多种呈现方式，而后端也由以前的Razor渲染HTML，转变为Stateless的RESTFulAPI，因此，我们需要一种标准的，通用的，无状态的，与语言无关的认证方式，也就是本文要介绍的****JwtBearer****认证。

JwtBearer是一种认证方式

## **Bearer认证**



在HTTP标准验证方案中，我们比较熟悉的是"Basic"和"Digest"，前者将用户名密码使用BASE64编码后作为验证凭证，后者是Basic的升级版，更加安全。

本文要介绍的****Bearer****验证也属于HTTP协议标准验证，它随着OAuth协议而开始流行，详细定义见： [RFC 6570](https://tools.ietf.org/html/rfc6750" \l "section-1.2)。

Bearer验证中的凭证称为BEARER\_TOKEN，或者是access\_token，Bearer验证的标准请求方式如下：

Authorization: Bearer [BEARER\_TOKEN]

## **JWT(JSON WEB TOKEN)**

上面介绍的Bearer认证，其核心便是****BEARER\_TOKEN****，而最流行的Token编码方式便是：JSON WEB TOKEN（JWT）。

JWT是由.分割的如下三部分组成：

### **头部(Header)**

Header 一般由两个部分组成：

alg是是所使用的hash算法，如：HMAC SHA256或RSA

typ是Token的类型，在这里就是：****JWT****。

{

"alg": "HS256",

"typ": "JWT"

}

然后使用Base64Url编码成第一部分：

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.<second part>.<third part>

### **载荷(Payload)**

这一部分是JWT主要的信息存储部分，其中包含了许多种的声明（claims）。

一个简单的Pyload可以是这样子的：

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

这部分同样使用Base64Url编码成第二部分：

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.<third part>

### **签名(Signature)**

Signature是用来验证发送者的JWT的同时也能确保在期间不被篡改。

在创建该部分时候你应该已经有了编码后的Header和Payload，然后使用保存在服务端的秘钥对其签名，一个完整的JWT如下：

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

## **示例**

### **模拟Token**

ASP.NET Core 内置的JwtBearer验证，并不包含Token的发放，我们先模拟一个简单的实现：

[HttpPost("authenticate")]

public IActionResult Authenticate([FromBody]UserDto userDto)

{

var user = \_store.FindUser(userDto.UserName, userDto.Password);

if (user == null) return Unauthorized();

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(Consts.Secret);

var authTime = DateTime.UtcNow;

var expiresAt = authTime.AddDays(7);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(JwtClaimTypes.Audience,"api"),

new Claim(JwtClaimTypes.Issuer,"http://localhost:5200"),

new Claim(JwtClaimTypes.Id, user.Id.ToString()),

new Claim(JwtClaimTypes.Name, user.Name),

new Claim(JwtClaimTypes.Email, user.Email),

new Claim(JwtClaimTypes.PhoneNumber, user.PhoneNumber)

}),

Expires = expiresAt,

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

var tokenString = tokenHandler.WriteToken(token);

return Ok(new

{

access\_token = tokenString,

token\_type = "Bearer",

profile = new

{

sid = user.Id,

name = user.Name,

auth\_time = new DateTimeOffset(authTime).ToUnixTimeSeconds(),

expires\_at = new DateTimeOffset(expiresAt).ToUnixTimeSeconds()

}

});

}

如上，使用微软提供的Microsoft.IdentityModel.Tokens帮助类(源码地址：[azure-activedirectory-identitymodel-extensions-for-dotnet](https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet))，可以很容易的创建出JwtToen，就不再多说。

### **注册JwtBearer认证**

首先添加JwtBearer包引用:

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 2.0.0

然后在Startup类中添加如下配置：

public void ConfigureServices(IServiceCollection services){

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(o =>

{

o.TokenValidationParameters = new TokenValidationParameters

{

NameClaimType = JwtClaimTypes.Name,

RoleClaimType = JwtClaimTypes.Role,

ValidIssuer = "http://localhost:5200",

ValidAudience = "api",

IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(Consts.Secret))

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TokenValidationParameters的参数默认值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// RequireSignedTokens = true,

// SaveSigninToken = false,

// ValidateActor = false,

// 将下面两个参数设置为false，可以不验证Issuer和Audience，但是不建议这样做。

// ValidateAudience = true,

// ValidateIssuer = true,

// ValidateIssuerSigningKey = false,

// 是否要求Token的Claims中必须包含Expires

// RequireExpirationTime = true,

// 允许的服务器时间偏移量

// ClockSkew = TimeSpan.FromSeconds(300),

// 是否验证Token有效期，使用当前时间与Token的Claims中的NotBefore和Expires对比

// ValidateLifetime = true

};

});

}

public void Configure(IApplicationBuilder app){

app.UseAuthentication();

}

在JwtBearerOptions的配置中，通常IssuerSigningKey(签名秘钥), ValidIssuer(Token颁发机构), ValidAudience(颁发给谁) 三个参数是必须的，后两者用于与TokenClaims中的Issuer和Audience进行对比，不一致则验证失败（与上面发放Token中的Claims对应）。

而NameClaimType和RoleClaimType需与Token中的ClaimType一致，在IdentityServer中也是使用的JwtClaimTypes，否则会造成User.Identity.Name为空等问题。

### **添加受保护资源**

创建一个需要授权的控制器，直接使用Authorize即可：

[Authorize]

[Route("api/[controller]")]

public class SampleDataController : Controller

{

[HttpGet("[action]")]

public IEnumerable<WeatherForecast> WeatherForecasts()

{

return ...

}

}

### **运行**

最后运行，直接访问/api/SampleData/WeatherForecasts，将返回一个401:

HTTP/1.1 401 Unauthorized

Server: Kestrel

Content-Length: 0

WWW-Authenticate: Bearer

让我们调用api/oauth/authenticate，获取一个JWT:

请求：

POST http://localhost:5200/api/oauth/authenticate HTTP/1.1

content-type: application/json

{

"username": "alice",

"password": "alice"

}

响应：

HTTP/1.1 200 OK

{"access\_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjEiLCJuYW1lIjoiYWxpY2UiLCJlbWFpbCI6ImFsaWNlQGdtYWlsLmNvbSIsInBob25lX251bWJlciI6IjE4ODAwMDAwMDAxIiwibmJmIjoxNTA5NDY0MzQwLCJleHAiOjE1MTAwNjkxNDAsImlhdCI6MTUwOTQ2NDM0MH0.Y1TDz8KjLRh\_vjQ\_3iYP4oJw-fmhoboiAGPqIZ-ooNc","token\_type":"Bearer","profile":{"sid":1,"name":"alice","auth\_time":1509464340,"expires\_at":1510069140}}

最后使用该Token，再次调用受保护资源：

GET http://localhost:5200/api/SampleData/WeatherForecasts HTTP/1.1

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjEiLCJuYW1lIjoiYWxpY2UiLCJlbWFpbCI6ImFsaWNlQGdtYWlsLmNvbSIsInBob25lX251bWJlciI6IjE4ODAwMDAwMDAxIiwibmJmIjoxNTA5NDY0MzQwLCJleHAiOjE1MTAwNjkxNDAsImlhdCI6MTUwOTQ2NDM0MH0.Y1TDz8KjLRh\_vjQ\_3iYP4oJw-fmhoboiAGPqIZ-ooNc

授权成功，返回了预期的数据：

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

[{"dateFormatted":"2017/11/3","temperatureC":35,"summary":"Chilly","temperatureF":94}]

## **扩展**

### **自定义Token获取方式**

JwtBearer认证中，默认是通过Http的Authorization头来获取的，这也是最推荐的做法，但是在某些场景下，我们可能会使用Url或者是Cookie来传递Token，那要怎么来实现呢？

.AddJwtBearer(o =>

{

o.Events = new JwtBearerEvents()

{

OnMessageReceived = context =>

{

context.Token = context.Request.Query["access\_token"];

return Task.CompletedTask;

}

};

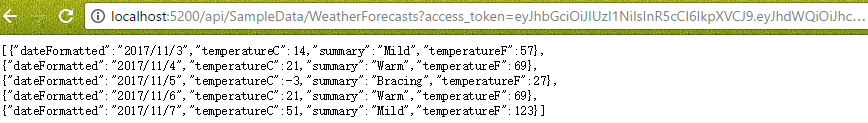
o.TokenValidationParameters = new TokenValidationParameters

{

...

};

然后在Url中添加access\_token=[token]，直接在浏览器中访问：



同样的，我们也可以很容易的在Cookie中读取Token，就不再演示。

除了OnMessageReceived外，还提供了如下几个事件：

****TokenValidated****：在Token验证通过后调用。

****AuthenticationFailed****: 认证失败时调用。

****Challenge****: 未授权时调用。