Creating And Validating JWT Tokens In ASP.NET Core

JANUARY 15, 2020 (HTTPS://DOTNETCORETUTORIALS.COM/2020/01/15/CREATING-AND-VALIDATING-JWT-TOKENS-IN-ASP-NET-CORE/) BY WADE (HTTPS://DOTNETCORETUTORIALS.COM/AUTHOR/ADMIN/) · 18 COMMENTS (HTTPS://DOTNETCORETUTORIALS.COM/2020/01/15/CREATING-AND-VALIDATING-JWT-TOKENS-IN-ASP-NET-CORE/#COMMENTS)

I've recently been using JWT Tokens as my authentication method of choice for my API's. And with it, I've had to do battle with various pieces of documentation on how JWT token authentication and authorization actually work in .NET Core.

Primarily, there is a lot of documentation on using ASP.NET Identity to handle authentication/authorization. So using the big bloated UserManager and using the packaged attributes like [Authorize] etc. However, I always get to a point where I just need a bit more custom flexibility, that the out of the box components don't provide. And when it comes to how to ***manually*** create JWT Tokens and validate them later on, the documentation is a little slim. Infact some guides show you how to manually create the token, but then tell you to use the out of the box components to validate it which creates confusion as to what you're actually doing. So here's hoping this article clears some things up!

Creating JWT Tokens In ASP.NET Core

Let's first take a look at how to create JWT tokens manually. For our example, we will simply create a service that returns a token as a string. Then however you return that token (header, response body etc) is up to you. I'll also note in the following examples, we have things like hardcoded “secrets”. I’m doing this for demonstration purposes but quite obviously you will want these to be config driven. You should take the following as a starting point, and then modify it to be production ready.

The code to generate a JWT Token looks like so :

```
public string GenerateToken(int userId)
{
        var mySecret = "asdv234234^&%&^%&^hjsdfb2%%%";
        var mySecurityKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes

        var myIssuer = "http://mysite.com";
        var myAudience = "http://myaudience.com";

        var tokenHandler = new JwtSecurityTokenHandler();
        var tokenDescriptor = new SecurityTokenDescriptor
        {
                Subject = new ClaimsIdentity(new Claim[]
                {
                        new Claim(ClaimTypes.NameIdentifier, userId.ToString
                }),
                Expires = DateTime.UtcNow.AddDays(7),
                Issuer = myIssuer,
                Audience = myAudience,
                SigningCredentials = new SigningCredentials(mySecurityKey, S
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
}
```

Let’s walk through this bit by bit.

I have a security key which is essentially used to “sign” the token on it’s way out. We can verify this signature when we receive the token on the other end to make sure it was created by us. Tokens themselves are actually readable even if you sign them so you should never put sensitive information in them. Signing simply verifies that it was us who created the token and whether it’s been tampered with, but it does not “encrypt” the token.

The Issuer and Audience are funny things because realistically, you probably won’t have a lot of use for them. Issuer is “who” created this token, for example your website, and Audience is “who” the token is supposed to be read by. So a good example might be that when a user logs in, your authentication api (auth.mywebsite.com) would be the issuer, but your general purposes API is the expected audience (api.mywebsite.com). These are actually free text fields so they don’t have to be anything in particular, but later on when we validate the issuer/audience, we will need to know what they are.

We are creating the token for 7 days, but you can set this to anything you want (Or have it not expire it at all), and the rest of the code is just .NET Core specific token writing code. Nothing too specific to what we are doing. Except for claims…

Explaining Claims

Claims are actually a simple concept, but too many articles go into the “abstract” thought process around them. In really simply terms, a claim is a “fact” stored in the token about the user/person that holds that token. For example, if I log into my own website as an administrator role, then my token might have a “claim” that my role is administrator. Or put into a sentence “Whoever holds this token can claim they are an admin”. That’s really what it boils down to. Just like you could store arbitrary information in a cookie, you can essentially do the same thing inside a JWT Token.

For example, because a claim “type” is simply a free text field, we can do things like :

```
Subject = new ClaimsIdentity(new Claim[]
{
        new Claim("UserRole", "Administrator"),
})
```

Notice how we don’t use the “ClaimTypes” static class like we did in the first example, we simply used a string to define the claim name, and then said what the claim value was. You can basically do this for any arbitrary piece of information you want, but again remember, anyone can decode the JWT Token so you should not be storing anything sensitive inside it.

I’ll also note that a great pattern to get into is to store the claim types as static consts/readonly. For example :

```
public static readonly string ClaimsRole = "UserRole";

[...]

Subject = new ClaimsIdentity(new Claim[]
{
        new Claim(ClaimsRole, "Administrator"),
})
```

You are probably going to need that ClaimType string in multiple places, so it’s better to set it once and reuse that static variable everywhere.

Validating A Token

So once you’ve created the token, the next step would be to validate it when a user sends you one. Now personally I like sending it inside a header like x-api-token, but because it’s simply a string, you can send it any which way you like. Because of that, let’s make our example method simply accept a token as a string and validate it.

```
public bool ValidateCurrentToken(string token)
{
        var mySecret = "asdv234234^&%&^%&^hjsdfb2%%%";
        var mySecurityKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes

        var myIssuer = "http://mysite.com";
        var myAudience = "http://myaudience.com";

        var tokenHandler = new JwtSecurityTokenHandler();
        try
        {
                tokenHandler.ValidateToken(token, new TokenValidationParamet
                {
                        ValidateIssuerSigningKey = true,
                        ValidateIssuer = true,
                        ValidateAudience = true,
                        ValidIssuer = myIssuer,
                        ValidAudience = myAudience,
                        IssuerSigningKey = mySecurityKey
                }, out SecurityToken validatedToken);
        }
        catch
        {
                return false;
        }
        return true;
}
```

You'll notice that I've had to copy and paste the security keys, issuer and audience into this method. As always, this would be better in a configuration class rather than being copied and pasted, but it makes the example a little easier to read.

So what's going on here? It's pretty simply actually. We create a TokenHandler which is a .NET Core inbuilt class for handling JWT Tokens, we pass it our token as well as our "expected" issuer, audience and our security key and call validate. This validates that the issuer and audience are what we expect, and that the token is signed with the correct key. An exception is thrown if the token is not validated so we can simply catch this and return false.

Reading Claims

So the final piece of the puzzle is reading claims. This is actually fairly easy assuming we have already validated the token itself.

```
public string GetClaim(string token, string claimType)
{
        var tokenHandler = new JwtSecurityTokenHandler();
        var securityToken = tokenHandler.ReadToken(token) as JwtSecurityToke

        var stringClaimValue = securityToken.Claims.First(claim => claim.Typ
        return stringClaimValue;
}
```

Read the token, go to the claims list, and find the claim with the matching type (remembering the claimType is simply a freetext string), and return the value.

What About AddAuthentication/AddJwtBearer?

So you might have read documentation that uses the following code :

```
services.AddAuthentication(x =>
{
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
        x.TokenValidationParameters = new TokenValidationParameters();
});
```

Or some variation with it that sets up the token validation parameters with signing keys, audiences and issuers. This only works if you are using the default Authorize attribute. These settings are a way for you to configure the inbuilt ASP.NET Core authorization handlers. **It does not set any global settings for JWT Tokens if you are creating/validating them yourself**.

Why do I point this out? I've seen people manually validating tokens and *not* validating the signing key. When I ask why they are not validating that the token is signed correctly, they have assumed that if they call AddJwtBearer with various settings that these also pass down anytime you call new JwtSecurityTokenHandler() . They do not!

Related Posts

1. **CSRF Tokens In ASP.NET Core (https://dotnetcoretutorials.com/2017/05/17/csrf-tokens-asp-net-core/)**
2. **CSRF Tokens In AngularJS/jQuery With ASP.NET Core (https://dotnetcoretutorials.com/2017/05/18/csrf-tokens-angularjsjquery-asp-net-core/)**
3. **Cookie Authentication In ASP.net Core (https://dotnetcoretutorials.com/2017/05/13/cookie-authentication-asp-net-core/)**
4. **Cookie Authentication In ASP.NET Core 2.0 (https://dotnetcoretutorials.com/2017/09/16/cookie-authentication-asp-net-core-2-0/)**

18 COMMENTS

Ali Kolahdoozan (http://www.iliasoft.ir)

January 15, 2020 at 4:05 pm (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-21858)

Can I ask a question ?.

What will happen if I copy the Token and Paste it in another Browser ?.

Reply

Wade

January 15, 2020 at 6:43 pm (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-21861)

It will work 100%. If that’s a negative for you, ask what would happen if you copied a cookie into another browser? Cookie or Tokens being stolen essentially amounts to the same thing, impersonation of another user. Theoretically you could do things like store an “IP” claim that is then validated on the server, but I’m not sure this would be a good idea.

Reply

Gary Mason

April 21, 2020 at 10:43 pm (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-25391)

If you specify the audience when creating the SecurityTokenDescriptor, at the point you generate a token, then hold a list of allowed audiences in your settings when you setup the TokenValidationParameters then you can limit where the token can be used. You can have a check when validating the token by getting the request scheme and host and checking it matches the token claims for the audience.

For example:

User requests a token and the audience in SecurityTokenDescriptor is set to :
$”{_accessor.HttpContext.Request.Scheme}://{_accessor.H

This will be the users scheme and host name, such as https://i.p (https://i.p) or hostname, you can even use the machine name if the i.p address can change.

When the token validation parameters are created you can sepcify the audiences allowed as a pre-defined list of allowed machine names or i.p addresses. This limits where the token can be used from.

Hope that makes sense.

Xavier Sarmiento C

January 29, 2020 at 4:57 pm (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-22253)

May you have the code because I've been trying to do something to use with JWT and Roles but it doesn't work, can you help me with that

Reply

Wade

January 30, 2020 at 7:07 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-22263)

If you are doing a completely custom implementation, you can just add a custom Claim with role and then check it on the filter.

Reply

dairiki

February 22, 2020 at 9:33 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-22797)

Dude this is a great POST! amazing thank you very much, I was looking for an example simple as this. The rest of the examples on the internet are just too complicated, this is exactly what I was looking for.

Reply

Will

March 11, 2020 at 3:43 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-23471)

This was exactly what I was looing for! Thanks so much!

Reply

Jane

April 29, 2020 at 11:47 pm (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-25609)

When do we call the ValidateToken() method? Also I'm receiving this error
EXCEPTION
Microsoft.IdentityModel.Tokens.SecurityTokenSignatureKeyNotFoundException:
IDX10500: Signature validation failed. No security keys were provided to validate the signature. at
System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateSignature(String token, TokenValidationParameters validationParameters) in C:\agent1_work\109\s\src\System.IdentityModel.Tokens.Jwt\JwtSecurityTokenHandler.cs:line 979
at System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateToken(String token, TokenValidationParameters validationParameters, SecurityToken& validatedToken) in C:\agent1_work\109\s\src\System.IdentityModel.Tokens.Jwt\JwtSecurityTokenHandler.cs:line 722
at Microsoft.Owin.Security.Jwt.JwtFormat.Unprotect(String protectedText)
at
Microsoft.Owin.Security.OAuth.OAuthBearerAuthenticationHandler.d__3.MoveNext()Diagnost
Context

Any clue why this occurs and how it can be fixed?

Reply

Wade

April 30, 2020 at 8:57 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-25619)

Are you passing in the correct security key?

So in this method here :

```
tokenHandler.ValidateToken(token, new TokenValidationParamete
{
        ValidateIssuerSigningKey = true,
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidIssuer = myIssuer,
        ValidAudience = myAudience,
        IssuerSigningKey = mySecurityKey //THIS
}, out SecurityToken validatedToken);
```

If you think you are, debug and make sure it's not blank etc.

Reply

Jane

April 30, 2020 at 6:49 pm (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-25626)

The correct security key is passed, the method returns true as well, but I still receive this error and I'm not quite sure where and when it is being thrown. So I wanted to confirm when is the right time to call ValidateCurrentToken() ?

Wade

May 1, 2020 at 8:56 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-25637)

How do you mean it returns true? If it returns true, then it's validated. Maybe throw up a quick POC on GIT and I can take a quick look for you. Or you can email the code to me at wade (at) dotnetcoretutorials.com

Pablo

June 8, 2020 at 9:23 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-26768)

Hello.
Why not to store the security key in the appsetings.json?
is it better your way?
kindly asked

Reply

Wade

June 8, 2020 at 11:18 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-26769)

The security key should be in some sort of config (Preferably something like KeyVault etc). However for this example I don't want to confuse where I'm pulling the configuration from so I just hardcoded it.

Reply

Bill

August 20, 2020 at 7:59 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-29550)

You should be able to validate against the application registered stuff by utilizing an IOptions class from the container for the configuration, something like this (untested):

```
    public class JwtTokenService : ITokenService
    {
        private readonly JwtBearerOptions _options;

        public JwtTokenService(IOptions options)
        {
            _options = options?.Value ?? throw new ArgumentNullException(nam
        }

        public bool ValidateToken(string token)
        {
            var parameters = _options.TokenValidationParameters;
            foreach (var tokenValidator in _options.SecurityTokenValidators)
            {
                try
                {
                    tokenValidator.ValidateToken(token, parameters, out _);
                }
                catch
                {
                    return false;
                }
            }

            return true;
        }
    }
```

Reply

David Arndt

September 10, 2020 at 5:33 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-30559)

I'm trying to validate a token created using Microsoft.Identity.Client in a mobile app. The token is passed in a header to a REST api. In the REST API, I need to validate the token. I've copied the example for Validating a Token (above) and substitued my own token, tenant ID, audience (app id), issuer and secret. It does not work. I get back this error:

kid: ‘System.String’.
Exceptions caught:
‘System.Text.StringBuilder’.
token: ‘System.IdentityModel.Tokens.Jwt.JwtSecurityToken’.
at System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateSignature(String token, TokenValidationParameters validationParameters)
at System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateToken(String token, TokenValidationParameters validationParameters, SecurityToken& validatedToken)

Can’t see what I’m doing wrong or differently except that perpaps the audience and/or the issuer might be wrong? For a mobile app, I thought these were supposed to be the app id and the tenant id, respectively.

I’m using this for the issuer:

$”https://login.microsoftonline.com/{myTenant}/v2.0";

And the audience is just my app id.

My “token” validates just fine here : https://jwt.ms/ (https://jwt.ms/)

Any ideas as to what might be wrong?

Reply

Wade

September 10, 2020 at 7:28 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-30563)

Are you sure that your token “validates” just fine at jwt.ms? Or it’s just read OK because any JWT token can be opened, and it’s weather it’s signed correctly is the question.

Just reading between the lines, it seems that you are using Azure B2C or similar as a login? That’s fine, but for those tokens I don’t believe you are supposed to validate them yourself, and you are supposed to use a library that will validate the token with the Microsoft API.

If you want to. Upload your code and email me at wade[at]dotnetcoretutorials.com and I’ll see if I can take a squiz.

Reply

Karen

September 14, 2020 at 10:28 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-30705)

Hi, thanks for the post, it is very simple. Just one question could you please also show an API controller example to understand how to authorize with this code?

Reply

josef

October 17, 2020 at 11:09 am (https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/#comment-31563)

This is a puzzle for me, I can’t join all the explained pieces, you have an example project, please

Reply

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Post Comment