



# Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 - Part 3

February 16, 2015 By [Taiseer Joudch](#) – 195 Comments

Be Sociable, Share!



This is the third part of Building Simple Membership system using ASP.NET Identity 2.1, ASP.NET Web API 2.2 and AngularJS. The topics we'll cover are:

- [Configure ASP.NET Identity with ASP.NET Web API \(Accounts Management\) – Part 1.](#)
- [ASP.NET Identity 2.1 Accounts Confirmation, and Password/User Policy Configuration – Part 2.](#)
- Implement JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – (This Post)
- [ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4](#)
- [ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)
- [AngularJS Authentication and Authorization with ASP.NET Web API and Identity 2.1 – Part 6](#)

The [source code](#) for this tutorial is available on GitHub.

## Implement JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1

Currently our API doesn't support authentication and authorization, all the requests we receive to any end point are done anonymously, In this post we'll configure our API which will act as our Authorization Server and Resource Server on the same time to issue JSON Web Tokens for authenticated users and those users will present this JWT to the protected end points in order to access it and process the request.



I will use step by step approach as usual to implement this, but I highly recommend you to read the post [JSON Web Token in ASP.NET Web API 2](#) before completing this one; where I cover deeply what is JSON Web Tokens, the benefits of using JWT over default access tokens, and how they can be used to decouple Authorization server from Resource server. In this tutorial and for the sake of keeping it simple; both OAuth 2.0 roles (Authorization Server and Resource Server) will live in the same API.

### Step 1: Implement OAuth 2.0 Resource Owner Password Credential Flow

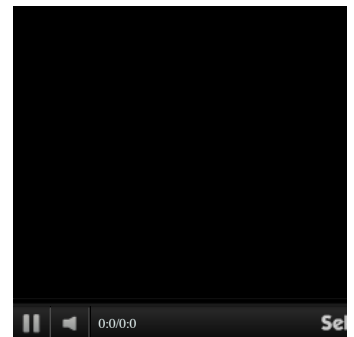
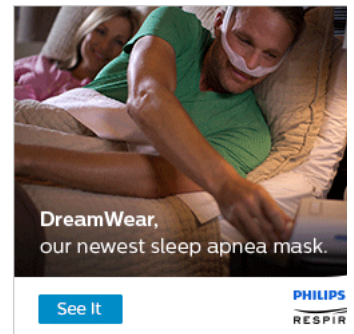
We are going to build an API which will be consumed by a trusted client (AngularJS front-end) so we only interested in implementing a single OAuth 2.0 flow where the registered user will present username and password to a specific end point, and the API will validate those credentials, and if all is valid it will return a JWT for the user where the client application used by the user should store it securely and locally in order to

#### ABOUT TAISEER



Father, MVP (ASP .NET/IIS), Scrum Master, Life Time Learner

#### CONNECT WITH ME



Advertise Here

Advertise Here

present this JWT with each request to any protected end point.

The nice thing about this JWT that it is a self contained token which contains all user claims and roles inside it, so there is no need to do any extra DB queries to fetch those values for the authenticated user. This JWT token will be configured to expire after 1 day of its issue date, so the user is requested to provide credentials again in order to obtain new JWT token.

If you are interested to know how to implement sliding expiration tokens and how you can keep the user logged in; I recommend you to read my other post [Enable OAuth Refresh Tokens in AngularJS App](#) which covers this deeply, but adds more complexity to the solution. To keep this tutorial simple we'll not add refresh tokens here but you can refer to the post and implement it.

To implement the Resource Owner Password Credential flow; we need to add new folder named "Providers" then add a new class named "CustomOAuthProvider", after you add then paste the code below:

```

1  public class CustomOAuthProvider : OAuthAuthorizationServerProvider
2  {
3
4      public override Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
5      {
6          context.Validated();
7          return Task.FromResult<object>(null);
8      }
9
10     public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
11     {
12
13         var allowedOrigin = "*";
14         context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { allowedOrigin });
15
16         var userManager = context.OwinContext.GetUserManager<ApplicationUserManager>();
17         ApplicationUser user = await userManager.FindAsync(context.UserName, context.Password);
18
19         if (user == null)
20         {
21             context.SetError("invalid_grant", "The user name or password is incorrect.");
22             return;
23         }
24
25         if (!user.EmailConfirmed)
26         {
27             context.SetError("invalid_grant", "User did not confirm email.");
28             return;
29         }
30
31         ClaimsIdentity oAuthIdentity = await user.GenerateUserIdentityAsync(userManager, "JWT");
32
33         var ticket = new AuthenticationTicket(oAuthIdentity, null);
34
35         context.Validated(ticket);
36
37     }
38 }
39
40

```

This class inherits from class "OAuthAuthorizationServerProvider" and overrides the below two methods:

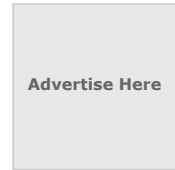
- As you notice the "ValidateClientAuthentication" is empty, we are considering the request valid always, because in our implementation our client (AngularJS front-end) is trusted client and we do not need to validate it.
- The method "GrantResourceOwnerCredentials" is responsible for receiving the username and password from the request and validate them against our ASP.NET 2.1 Identity system, if the credentials are valid and the email is confirmed we are building an identity for the logged in user, this identity will contain all the roles and claims for the authenticated user, until now we didn't cover roles and claims part of the tutorial, but for the mean time you can consider all users registered in our system without any roles or claims mapped to them.
- The method "GenerateUserIdentityAsync" is not implemented yet, we'll add this helper method in the next step. This method will be responsible to fetch the authenticated user identity from the database and returns an object of type "ClaimsIdentity".
- Lastly we are creating an Authentication ticket which contains the identity for the authenticated user, and when we call "context.Validated(ticket)" this will transfer this identity to an OAuth 2.0 bearer access token.



JavaScript Diag



Click here for C



## RECENT POSTS

[ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)

[ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4](#)

[Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3](#)

[ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration – Part 2](#)

[Interview with John about establishing a successful blog](#)

## BLOG ARCHIVES

Blog Archives

Select Month

## LEAVE YOUR EMAIL AND KEEP TUNED!

Sign up to receive email updates on every new post!

Email Address

SUBSCRIBE

## Step 2: Add method "GenerateUserIdentityAsync" to "ApplicationUser" class

Now we'll add the helper method which will be responsible to get the authenticated user identity (all roles and claims mapped to the user). The "userManager" class contains a method named "CreateIdentityAsync" to do this task, it will basically query the DB and get all the roles and claims for this user, to implement this open class "ApplicationUser" and paste the code below:

```
1 //Rest of code is removed for brevity
2 public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager, string authentication
3 {
4     var userIdentity = await manager.CreateIdentityAsync(this, authenticationType);
5     // Add custom user claims here
6     return userIdentity;
7 }
```

## Step 3: Issue JSON Web Tokens instead of Default Access Tokens

Now we want to configure our API to issue JWT tokens instead of default access tokens, to understand what is JWT and why it is better to use it, you can refer back to this [post](#).

First thing we need to installed 2 NueGet packages as the below:

```
1 Install-package System.IdentityModel.Tokens.Jwt -Version 4.0.1
2 Install-package Thinkecture.IdentityModel.Core -Version 1.3.0
```

There is no direct support for issuing JWT in ASP.NET Web API, so in order to start issuing JWTs we need to implement this manually by implementing the interface "ISecureDataFormat" and implement the method "Protect".

To implement this add new file named "CustomJwtFormat" under folder "Providers" and paste the code below:

```
1 public class CustomJwtFormat : ISecureDataFormat<AuthenticationTicket>
2 {
3     private readonly string _issuer = string.Empty;
4     public CustomJwtFormat(string issuer)
5     {
6         _issuer = issuer;
7     }
8     public string Protect(AuthenticationTicket data)
9     {
10         if (data == null)
11         {
12             throw new ArgumentNullException("data");
13         }
14         string audienceId = ConfigurationManager.AppSettings["as:AudienceId"];
15         string symmetricKeyAsBase64 = ConfigurationManager.AppSettings["as:AudienceSecret"];
16         var keyByteArray = TextEncodings.Base64Url.Decode(symmetricKeyAsBase64);
17         var signingKey = new HmacSigningCredentials(keyByteArray);
18         var issued = data.Properties.IssuedUtc;
19         var expires = data.Properties.ExpiresUtc;
20         var token = new JwtSecurityToken(_issuer, audienceId, data.Identity.Claims, issued.Value.UtcDateTime, expires, signingKey);
21         var handler = new JwtSecurityTokenHandler();
22         var jwt = handler.WriteToken(token);
23         return jwt;
24     }
25     public AuthenticationTicket Unprotect(string protectedText)
26     {
27         throw new NotImplementedException();
28     }
29 }
```

What we've implemented in this class is the following:

- The class “CustomJwtFormat” implements the interface “ISecureDataFormat<AuthenticationTicket>”, the JWT generation will take place inside method “Protect”.
- The constructor of this class accepts the “Issuer” of this JWT which will be our API. This API acts as Authorization and Resource Server on the same time, this can be string or URI, in our case we’ll fix it to URI.
- Inside “Protect” method we are doing the following:
  - As we stated before, this API serves as Resource and Authorization Server at the same time, so we are fixing the Audience Id and Audience Secret (Resource Server) in web.config file, this Audience Id and Secret will be used for HMAC265 and hash the JWT token, I’ve used this [implementation](#) to generate the Audience Id and Secret.
  - Do not forget to add 2 new keys “as:AudienceId” and “as:AudienceSecret” to the web.config AppSettings section.
  - Then we prepare the raw data for the JSON Web Token which will be issued to the requester by providing the issuer, audience, user claims, issue date, expiry date, and the signing key which will sign (hash) the JWT payload.
  - Lastly we serialize the JSON Web Token to a string and return it to the requester.
- By doing this, the requester for an OAuth 2.0 access token from our API will receive a signed token which contains claims for an authenticated Resource Owner (User) and this access token is intended to certain (Audience) as well.

## Step 4: Add Support for OAuth 2.0 JWT Generation

Till this moment we didn’t configure our API to use OAuth 2.0 Authentication workflow, to do so open class “Startup” and add new method named “ConfigureOAuthTokenGeneration” as the below:

```

1 private void ConfigureOAuthTokenGeneration(IApplicationBuilder app)
2 {
3     // Configure the db context and user manager to use a single instance per request
4     app.CreatePerOwinContext(ApplicationDbContext.Create);
5     app.CreatePerOwinContext(ApplicationUserManager.Create);
6
7     OAuthAuthorizationServerOptions OAuthServerOptions = new OAuthAuthorizationServerOptions()
8     {
9         //For Dev environment only (on production should be AllowInsecureHttp = false)
10        AllowInsecureHttp = true,
11        TokenEndpointPath = new PathString("/oauth/token"),
12        AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),
13        Provider = new CustomOAuthProvider(),
14        AccessTokenFormat = new CustomJwtFormat("http://localhost:59822")
15    };
16
17    // OAuth 2.0 Bearer Access Token Generation
18    app.UseOAuthAuthorizationServer(OAuthServerOptions);
19 }

```

What we’ve implemented here is the following:

- The path for generating JWT will be as :”http://localhost:59822/oauth/token”.
- We’ve specified the expiry for token to be 1 day.
- We’ve specified the implementation on how to validate the Resource owner user credential in a custom class named “CustomOAuthProvider”.
- We’ve specified the implementation on how to generate the access token using JWT formats, this custom class named “CustomJwtFormat” will be responsible for generating JWT instead of default access token using DPAPI, note that both format will use **Bearer** scheme.

Do not forget to call the new method “ConfigureOAuthTokenGeneration” in the Startup “Configuration” as the class below:

```

1 public void Configuration(IApplicationBuilder app)
2 {
3     HttpConfiguration httpConfig = new HttpConfiguration();
4
5     ConfigureOAuthTokenGeneration(app);
6
7     //Rest of code is removed for brevity
8
9 }

```

Our API currently is ready to start issuing JWT access token, so test this out we can issue HTTP POST

## BIT OF TECHNOLOGY

[ARCHIVE](#)[ABOUT ME](#)[SPEAKING](#)[CONTACT](#)

form-data x-www-form-urlencoded raw binary

username SuperPowerUser

password MySuperP@ssl

grant\_type password

Key Value

Send Preview Pre-request script Tests Add to collection

Body Cookies Headers (10) Tests STATUS 200 OK TIME 2690

Pretty Raw Preview JSON

```
{
  "access_token":
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0eW1laWQioiI...
    ouvknK3erDc",
  "token_type": "bearer",
  "expires_in": 86399
}
```

### Step 5: Protect the existing end points with [Authorize] Attribute

Now we'll visit all the end points we have created earlier in previous posts in the "AccountsController" class, and attribute the end points which need to be protected (only authenticated user with valid JWT access token can access it) with the [Authorize] attribute as the below:

– **GetUsers, GetUser, GetUserByName, and DeleteUser** end points should be accessed by users enrolled in Role "Admin". Roles Authorization is not implemented yet and for now we will only allow any authentication user to access it, the code change will be as simple as the below:

```
1 [Authorize]
2 [Route("users")]
3 public IHttpActionResult GetUsers()
4 {}
5
6 [Authorize]
7 [Route("user/{id:guid}", Name = "GetUserById")]
8 public async Task<IHttpActionResult> GetUser(string id)
9 {}
10
11
12 [Authorize]
13 [Route("user/{username}")]
14 public async Task<IHttpActionResult> GetUserByName(string username)
15 {
16 }
17
18 [Authorize]
19 [Route("user/{id:guid}")]
20 public async Task<IHttpActionResult> DeleteUser(string id)
21 {
22 }
```

– **CreateUser and ConfirmEmail** endpoints should be accessed anonymously always, so we need to attribute it with [AllowAnonymous] as the below:

```
1 [AllowAnonymous]
2 [Route("create")]
```

```

3 public async Task<IHttpActionResult> CreateUser(CreateUserBindingModel createUserModel)
4 {
5 }
6
7 [AllowAnonymous]
8 [HttpGet]
9 [Route("ConfirmEmail", Name = "ConfirmEmailRoute")]
10 public async Task<IHttpActionResult> ConfirmEmail(string userId = "", string code = "")
11 {
12 }

```

– **ChangePassword** endpoint should be accessed by the authenticated user only, so we'll attribute it with **[Authorize]** attribute as the below:

```

1 [Authorize]
2 [Route("ChangePassword")]
3 public async Task<IHttpActionResult> ChangePassword(ChangePasswordBindingModel model)
4 {
5 }

```

## Step 6: Consume JSON Web Tokens

Now if we tried to obtain an access token by sending a request to the end point “oauth/token” then try to access one of the protected end points we'll receive 401 Unauthorized status, the reason for this that our API doesn't understand those JWT tokens issued by our API yet, to fix this we need to the following:

Install the below NuGet package:

```

1 Install-Package Microsoft.Owin.Security.Jwt -Version 3.0.0

```

The package “Microsoft.Owin.Security.Jwt” is responsible for protecting the Resource server resources using JWT, it only validate and de-serialize JWT tokens.

Now back to our “Startup” class, we need to add the below method “ConfigureOAuthTokenConsumption” as the below:

```

1 private void ConfigureOAuthTokenConsumption(IAppBuilder app) {
2
3     var issuer = "http://localhost:59822";
4     string audienceId = ConfigurationManager.AppSettings["as:AudienceId"];
5     byte[] audienceSecret = TextEncodings.Base64Url.Decode(ConfigurationManager.AppSettings["as:AudienceSecret"]);
6
7     // Api controllers with an [Authorize] attribute will be validated with JWT
8     app.UseJwtBearerAuthentication(
9         new JwtBearerAuthenticationOptions
10         {
11             AuthenticationMode = AuthenticationMode.Active,
12             AllowedAudiences = new[] { audienceId },
13             IssuerSecurityTokenProviders = new IIssuerSecurityTokenProvider[]
14             {
15                 new SymmetricKeyIssuerSecurityTokenProvider(issuer, audienceSecret)
16             }
17         });
18 }

```

This step will configure our API to trust tokens issued by our Authorization server only, in our case the Authorization and Resource Server are the same server (http://localhost:59822), notice how we are providing the values for audience, and the audience secret we used to generate and issue the JSON Web Token in step3.

By providing those values to the “JwtBearerAuthentication” middleware, our API will be able to consume only JWT tokens issued by our trusted Authorization server, any other JWT tokens from any other Authorization server will be rejected.

Lastly we need to call the method “ConfigureOAuthTokenConsumption” in the “Configuration” method as the below:

```

1 public void Configuration(IAppBuilder app)
2 {
3     HttpConfiguration httpConfig = new HttpConfiguration();
4

```



```
5      ConfigureOAuthTokenGeneration(app);
6
7      ConfigureOAuthTokenConsumption(app);
8
9      //Rest of code is here
10
11 }
```

## Step 7: Final Testing

All the pieces should be in place now, to test this we will obtain JWT access token for the user “SuperPowerUser” by issuing POST request to the end point “oauth/token”

Token (Password Grant)

http://localhost:59822/oauth/token

POST

URL params

Headers (2)

Accept

application/json

Content-Type

application/json

Header

Value

Add preset

Manage presets

form-data

x-www-form-urlencoded

raw

binary

username

SuperPowerUser

password

MySuperP@ss!

grant\_type

password

Key

Value

Send

Save

Preview

Pre-request script

Tests

Add to collection

Reset

Body

Cookies

Headers (10)

Tests

STATUS 200 OK

TIME 28174 ms

Pretty

Raw

Preview

JSON

Copy

```
{
  "access_token":
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0eXlwIiwiaWF0IjIyMDYyZW00OWwLTQ5YyLvaTrpMss-xOCtQM",
  "token_type": "bearer",
  "expires_in": 86399
}
```

Then we will use the JWT received to access protected end point such as “ChangePassword”, if you remember once we added this end point, we were not able to test it directly because it was anonymous and inside its implementation we were calling the method “User.Identity.GetUserId()”. This method will return nothing for anonymous user, but after we’ve added the [Authorize] attribute, any user needs to access this end point should be authenticated and has a valid JWT.

To test this out we will issue POST request to the end point “/accounts/ChangePassword” as the image below, notice here we are setting the **Authorization header** using **Bearer** scheme setting its value to the JWT we received for the user “SuperPwoerUser”. If all is valid we will receive 200 OK status and the user password should be updated.

Change Password

http://localhost:59822/api/accounts/changepassword POST URL params Headers (3)

Accept application/json

Content-Type application/json

Authorization Bearer eyJ0eXAiOiJKV1QiLCJhbG

Header Value

Add preset Manage presets

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "OldPassword": "MySuperP@ss!",
3   "NewPassword": "MySuperP@ss!",
4   "ConfirmPassword": "MySuperP@ss!"
5 }
6
```

Send Save Preview Pre-request script Tests Add to collection Reset

The **source code** for this tutorial is available on [GitHub](#).

In the **next post** we'll see how we'll implement Roles Based Authorization in our Identity service.

Follow me on Twitter [@tjoudeh](#)

## References

- [Understanding OWIN/Katana Authentication/Authorization Part I: Concepts](#) by John Atten
- [Featured Image Source](#)

Be Sociable, Share!



Like this:



Be the first to like this.

## Related Posts

[ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)

[ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4](#)

[AngularJS Authentication Using Azure Active Directory Authentication Library \(ADAL\)](#)

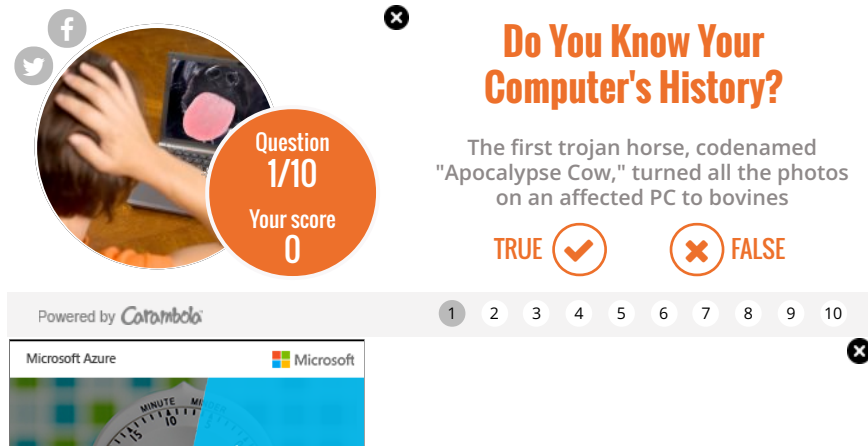
[JSON Web Token in ASP.NET Web API 2 using Owin](#)

[Decouple OWIN Authorization Server from Resource Server](#)

Filed Under: [ASP.NET](#), [ASP.NET Identity](#), [ASP.Net Web API](#), [CodeProject](#), [Web API Security](#), [Web API Tutorial](#)

Tagged With: [Authentication](#), [Authorization](#), [JSON Web Tokens](#), [JWT](#), [OAuth](#)





**Do You Know Your Computer's History?**

The first trojan horse, codenamed "Apocalypse Cow," turned all the photos on an affected PC to bovines

Question 1/10  
Your score 0

TRUE ☒ FALSE ☒

Powered by Carambola

Microsoft Azure Microsoft

## Comments



Jacob Cheriathundam says  
August 22, 2015 at 8:27 am

Thank you so much for the tutorial Taiseer! You explain an incredibly difficult topic in very easy terms.

I did run into one problem after tying all this in to a sample Angular App that I was building. Every so often, I'll get this error "IDX 10223: Lifetime Validation Failed" that is tied to the JWT token (I'm assuming). My guess is that it's an expired token on the server but somehow I need to catch this error in my client code. I wanted to track down where in my code it's actually happening but my browsers start acting weird after I get the error. Chrome crashes if I open dev tools after getting this error (oddly enough, if I don't open dev tools, the entire app works just fine). Visual Studio will throw an error from the code saying that the IDX error was encountered, but IE won't let me navigate back to the code. Firefox Dev Edition throws, what seems to be, future errors that would occur if the token expired but doesn't show me where the actual code is erroring out.

Do you have any thoughts on how to handle this error? Should I be catching it in my authentication code? Should I put an httpinterceptor in place in Angular that monitors for this error (I'm assumign its a 401 error ultimately)?

Any help would be greatly appreciated!

Reply



Taiseer Joudeh says  
August 24, 2015 at 11:00 pm

Hi Jacob,

That is really interesting issue, well if I were in your case, then I will try using http interceptor where I will be monitoring 401 response. But in any case if the JWT is expired the server should never throw an exception, it should return 401 only. To make sure of this, please try to use expired JWT in a request to protected resource using PostMan and check the response, it should only return 401. Please share with me your solution, hope it will be an easy one.

Reply



alexismeillandAlexis says  
August 23, 2015 at 8:08 pm

Hi,

I have followed your tutorial. It's awesome. But I have an issue.

When I try to Change the password. I pass the bearer token.

And I have a notimplemented exception on JwtFormat class in provider folder.

```
public AuthenticationTicket Unprotect(string protectedText)
```

Why is it going in this method?

How can I fix it?

Thanks,

[Reply](#)



**Taiseer Joudeh** says

August 24, 2015 at 10:53 pm

Hi Alex,

I'm not sure if you implemented something incorrectly in this [class](#), but are you receiving this exception for all protected endpoints or just this one?

[Reply](#)



**alex** says

August 25, 2015 at 12:06 am

Hi,

Thanks I could fix it. The nugget Microsoft.Owin.Security.Jwt -Version 3.0.0 was not installed.

Thanks!

[Reply](#)



**alexismeilland** says

August 29, 2015 at 3:06 am

Hi,

I can create my token with postman and it works perfectly but with my app it does not work.I have the error:

XMLHttpRequest cannot load <http://localhost:7241/oauth/token>. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:9000' is therefore not allowed access. The response had HTTP status code 400.

I have enabled cors in my web api.

And in my angularjs app.js file:

```
$httpProvider.defaults.useXDomain = true;  
delete $httpProvider.defaults.headers.common['X-Requested-With'];  
// $httpProvider.defaults.withCredentials = true;  
$httpProvider.interceptors.push('AuthInterceptor');
```

I cannot sort it out. Any ideas?

I tried with chrome and firefox.

Thanks,

[Reply](#)



Taiseer Joudeh says

September 7, 2015 at 6:52 am

Hi,

you need to allow it here [too](#) As well make sure you are using the same NuGet versions I'm using.

[Reply](#)



Avi says

September 8, 2015 at 2:50 am

Hi Taiseer, This absolutely awesome. I do have a question, let's say a third party untrusted client is trying to accessing the web api. So obviously, I don't want the user to create their account and authenticate via the third party client, how can I make sure third party request redirects to my webapi backend, so that they can authenticate on my webapi and then go back to the third party client once done.

[Reply](#)



Taiseer Joudeh says

September 10, 2015 at 1:05 pm

Hi Avi,

You need to use a different grant here which is the implicit flow or authorization code flow, where the user will enter his credentials not in the untrusted client, but directly in the authorization server. I didn't implement this but [this post](#) will help you to achieve what you are looking for, or you can consider using the [ThinkTecture Identity Server](#).

[Reply](#)



Avi says

September 10, 2015 at 9:23 pm

Thanks Taiseer. I'm thinking of looking at the Thinktecture Id server.

[Reply](#)



Jeroen-bart Engelen says

September 10, 2015 at 6:29 pm

Great article, but I'm having trouble getting it to work.  
I've made a test project based on this article and this

(<http://odetocode.com/blogs/scott/archive/2015/01/15/using-json-web-tokens-with-katana-and-webapi.aspx>) article. The project successfully creates a JWT token, but I cannot perform any requests using that token. I keep getting an HTTP 401 error. The problem is that I have no idea how to debug this inner process. It's all handled by the Authorize attribute. Do you have any pointers how to troubleshoot this problem?

Thanks!

Reply



Taiseer Joudeh says

September 10, 2015 at 6:46 pm

Hi Jeroen, this issued JWT token should be trusted by your resource API, please check step 6 from [my post](#) and make sure that AudienceId, secret and issuer are all set correctly.

Reply



Jeroen-bart Engelen says

September 10, 2015 at 9:04 pm

After building the complete Owin framework from source to be able to debug it, I was able to solve it. I'm using Web API and I want my API calls to be authorized. A call with bearer token was denied constantly, even though I was able to get an OAuth token.

I put a breakpoint on `AuthenticateCoreAsync()` in `OAuthBearerAuthenticationHandler` from the Microsoft Owin security library I noticed it wouldn't get hit. At all. As if the authentication middleware wasn't loaded. I checked my implementation 10 times and it all seemed good. Per chance I requested the index page (which is just an empty directory) and then the breakpoint was hit! But for some reason, requesting the Web API endpoint didn't work.

I then changed the order in which I loaded the middleware from:

```
app.UseWebApi(config);
app.UseJwtBearerAuthentication(new JwtOptions());
app.UseOAuthAuthorizationServer(new OAuthOptions());
```

to:

```
app.UseJwtBearerAuthentication(new JwtOptions());
app.UseOAuthAuthorizationServer(new OAuthOptions());
app.UseWebApi(config);
```

and then it worked....

I would've never thought the order would be important....

Reply



Achilleas says

March 7, 2016 at 5:37 pm

The order is indeed important! I was getting 401 too, and then changed the order from:

```
HttpConfiguration httpConfig = new HttpConfiguration();
```

```
ConfigureOAuthTokenGeneration(app);
```

```
ConfigureWebApi(httpConfig);
```

```
app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
```

```
app.UseWebApi(httpConfig);
```

```
ConfigureOAuthTokenConsumption(app);
```

To:

```
HttpConfiguration httpConfig = new HttpConfiguration();
```

```
ConfigureOAuthTokenConsumption(app);
```

```
ConfigureOAuthTokenGeneration(app);
```

```
ConfigureWebApi(httpConfig);
```

```
app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
```

```
app.UseWebApi(httpConfig);
```

and it worked!

Thank you so much for finding that out, you really saved hours of work for such a silly problem!

[Reply](#)



Carlos Boichuk says

September 13, 2015 at 11:27 pm

Taiseer, this is an excellent article, I have a question about how to customize the json web token consumption in a web api resource server, i want to not only validate the json web token receive on each request also i want to read the token received, which methods should i override in the web api resource server in order to perform the current json web token validation as the one you mentioned in this article and in the others, and implementing this kind of additional logic to it ?

Thanks for your articles, ill be waiting for your reply.

[Reply](#)



Taiseer Joudeh says

September 18, 2015 at 2:50 pm

Hi Carols, I need to check this, if you find an answer please share it.

[Reply](#)



Michael Heribert says

September 20, 2015 at 11:08 am

Hi, thank you for the great tutorial, it helped me a lot! Could you pls explain how exactly you implemented the as:AudienceId and Secret? How can I create the values for my application? And when using the function linked in the description, how should it be implemented into the application?

thanks in advance 😊

[Reply](#)



Michael Heribert says

September 20, 2015 at 8:59 pm

Hi, I managed to solve the problem: using the x-www-form-urlencoded option for the body produces an error, using the raw tab instead solved the problem whereas I expected the audience-key to be the source of the problem.

greetings

Michael Heribert

[Reply](#)



Taiseer Joudeh says

October 3, 2015 at 5:51 pm

You are welcome Michael, thanks for you comment and happy to hear you solved the issue.

[Reply](#)



Manuel Espino says

December 15, 2015 at 7:08 pm

Hi Michael

I'm trying to create my own audience id and secret, could you please tell me how you managed to create your own?

Thanks in advace!

[Reply](#)



Russell Wyatt says

March 5, 2016 at 11:25 am

I would like an answer to this too. It's a great tutorial but this bit has left me a bit high and dry. Am I to build this class into another application to generate the values? Can you expand on this please?

TIA

[Reply](#)



mbeddedsoft says

September 21, 2015 at 12:17 am

Getting error testing out access token. The error is unsupported\_grant\_type.

Request: <http://localhost:59822/oauth/token>

Status

400 Bad Request Show explanation Loading time: 93



## Request headers

username: SuperPowerUser

Origin: chrome-extension://hgmloofddfnphfgcellkdfbfjeloo

X-DevTools-Emulate-Network-Conditions-Client-Id: 186AD051-64D8-4890-9BBE-47F7B6309430

grant\_type: password

User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93

Safari/537.36

password: MySuperP@ssword!

Content-Type: application/x-www-form-urlencoded

Accept: \*/\*

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.8

Cookie: \_\_RequestVerificationToken\_L2JvcmlRlcnpA2=qA73L3FxlS562JBDGKJR-ch6bnrrlW-

zFqx5ntdDAZPZpl7C9nOAoj4uRvDKPNYXepRSZJHOwixnlmp9forV22k0kn-atcTlfhi8AnJ4fwk1;

\_\_RequestVerificationToken=AqounhGRCOyT\_drnSmV\_alsJQ8Nl0eRJkwECTarf079luqmi9AQVaeEXUMQ\_0WH6B8KQW\_j7lq-

H5FxBUPWZqRPaEXTNNfslOmdTDfzG4U1

## Response headers

Cache-Control: no-cache

Pragma: no-cache

Content-Length: 34

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

Expires: -1

Server: Microsoft-IIS/8.0

X-SourceFiles: =?UTF-8?B?

YzpcdXNlcnNcbWFya1xkb2NlbWVudHNcdmlzdWFsIHNoZWVudHJpYyAyMDEzXFB5b2plY3RzXFdlYkFwcDFcV2ViQXBwMVxvYXV0aFw0b2tldG==?

=

X-Powered-By: ASP.NET

Date: Sun, 20 Sep 2015 21:08:55 GMT

Raw

JSON

```
{ "error": "unsupported_grant_type" }
```

I did some research and most of the stuff I found said to add the existing line in

GrantResourceOwnerCredentials()...

```
context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "*" });
```

Since this is already in CustomOAuthProvider.GrantResourceOwnerCredentials(), I'm kinda stuck.

Do you have any ideas?

thanks,

[Reply](#)



pepek says

November 20, 2015 at 5:48 pm

Hi. I guess username, passwod and grant\_type should be sent in body not in headers

[Reply](#)



Taiseer Joudh says

November 24, 2015 at 1:34 pm

Hi Pepek, Nothing sent in the headers, it is sent in the body as UrlEncoded content type.

[Reply](#)



mbeddedsoft says

September 21, 2015 at 5:09 pm

I had a problem when I accidentally updated packages to latest versions for \*.owin.\* in PM. I meant to only update EF and the .net framework to 4.5.1. After updating I noticed that the method 'public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)' was not being called. I set a breakpoint in the method and it never hit when I tried making a REST client to call POST <http://localhost:port/oauth/token> with username, password, and grant\_type params. Reverting back to the 2.0 versions identified in article 1 fixed this.

Would you know why this would be? Just asking.

BTW, I am really enjoying your article. Good information. I also really appreciate all the referenced links and articles. You are doing a fantastic job. It's good to be able to follow the tutorial and also have references to more detailed information on Identity 2.0 and OWIN.

thank you so much.

[Reply](#)



Prashanth says

October 11, 2015 at 7:41 am

Hi mbeddedsoft,  
Even I am facing same problem. Could u plz explain what have u done to solve this. Plz give more info about which package needs to update ?

[Reply](#)



V says

September 22, 2015 at 5:00 pm

Great post. By registering client id with Authorization server, and securing the JWT token by signing and encryption, are we achieving the OpenIDConnect 2.0 specifications. Can you please clarify.

[Reply](#)



Daniel says

September 30, 2015 at 5:38 pm

Hi Taiseer,

Many thanks for your article. It is very interesting and helpful for someone like me who's not a server expert 😊

I learnt a lot from your article and I've done all the steps. But I've got an issue that you don't seem to have, and I don't know why.

1. I can generate IDs and Secrets, no problem, and also request the token. But the response (containing the token) is not properly formatted. PostMan says there's no headers, and the characters are unknown. Do you have an idea ?

2. Bonus question : Which function is hit when I call the URL for retrieving the token (localhost:xxxxx/oauth/token) ? I would like to do some debugging for 1., but my breakpoints don't stop the flow.

Thanks a lot !

[Reply](#)



Taiseer Joudeh says

October 3, 2015 at 6:11 pm

Hi Daniel,

For your second question you can put breakpoints in methods "ValidateClientAuthentication" and "GrantResourceOwnerCredentials" in the class which inherits from "OAuthAuthorizationServerProvider".

For your first one I didn't really understand it, can you elaborate more please?

Regards,

Taiseer

[Reply](#)



iliya says

October 8, 2015 at 7:31 pm

Hi Taiseer!

thanks very much, it's best solution for authentication and authorization in web api.

i have some important questions :

1.in step 3 inside "CustomJwtFormat" and step 6 inside "ConfigureOAuthTokenConsumption", you used static value from app setting for sample, is it true? but in real Project we can't do that because audience id and audience secret are different for each user... so what should we do?

2.in step 6 inside "ConfigureOAuthTokenConsumption" how can i read JWT tokens that sent clients with their requests?

instead this code :

```
string audienceId = ConfigurationManager.AppSettings["as:AudienceId"];
byte[] audienceSecret =
TextEncodings.Base64Url.Decode(ConfigurationManager.AppSettings["as:AudienceSecret"]);
```

and how should i check user's token with the user?

[Reply](#)



Mike Olson says

October 9, 2015 at 10:16 pm

I have implemented this tutorial, but when I secure my WebAPI methods with the AuthorizeAttribute they become unavailable, and nothing actually consumes the Jwt Token. There are no errors or anything, so I'm not sure what I'm missing. I can see that the token is created, it is passed back to Angular, and it is included in subsequent requests from Angular to the WebAPI. I can even write a custom AuthorizeAttribute, and if I break in there I can see that the token is in the header of the request. I'm going crazy trying to figure this out. Can you help me?

[Reply](#)



Taiseer Joudeh says

October 12, 2015 at 10:19 am

Hi Mike,  
What do you mean by the become unavailable? You mean Unauthorized?  
You need to make sure that issuer and audience id are exactly the same in Resource API, use JWT.IO to check your JWT token and inspect the claims inside it.  
Hope this will help.

Reply



Mike Olson says  
October 12, 2015 at 5:40 pm

Sorry I should have been more clear...

The JWT token is present in the request, but no ClaimsIdentity is created (that I can so), and just overall I can't find any information about the user context based on what was in the JWT token. The one thing I did differently from your tutorial was to skip all the stuff with the UserManager, since I have a previously existing user database that I'm hooking into, so I wonder if that's part of the problem, but it didn't seem like it should be.

I did wonder if it was because the issuer and audience IDs weren't the same, but they are. I checked using JWT.IO and I even made them string constants in the code, and the code that issues uses the same constants as the code that calls UseJwtBearerAuthentication.

Reply



Mike Olson says  
October 13, 2015 at 10:58 pm

I figured out what I was doing wrong. I put app.UseWebApi(config) at the top of the Startup code instead of the bottom. So it was sending me to WebAPI before the authorization modules were ever getting called. I asked the question a little better over at the ASP.NET forums and got an answer:

<http://forums.asp.net/p/2070482/5976299.aspx?p=True&t=635803483826595456>

Reply



Senthil says  
October 23, 2015 at 7:26 pm

Thank you so much for the wonderful article.

I created two separate API projects for AS & RS.

Token created from AS is perfectly working as expected in RS.

But when I use the same token in AS, it goes to Unprotect(string protectedText)

I saw the first comment with same case, but I updated Microsoft.Owin.Security.Jwt 3.0.1.0

AS having other Controllers without Authorize and I am able to get result from those.

If I use Authorize Bearer in AS – it goes to Unprotect method.

Here is my CustomJwtFormat class:

```

public class CustomJwtFormat : ISecureDataFormat
{
    private readonly string _issuer = string.Empty;
    private const string AudiencePropertyKey = "audience";

    public CustomJwtFormat(string issuer)
    {
        _issuer = issuer;
    }

    public string Protect(AuthenticationTicket data)
    {
        if (data == null)
        {
            throw new ArgumentNullException("data");
        }

        string audienceId = data.Properties.Dictionary.ContainsKey(AudiencePropertyKey) ?
            data.Properties.Dictionary[AudiencePropertyKey] : null;

        if (string.IsNullOrEmpty(audienceId)) throw new InvalidOperationException("AuthenticationTicket.Properties
            does not include audience");

        Audience audience = AudienceManager.FindAudience(audienceId);

        string symmetricKeyAsBase64 = audience.Base64Secret;

        //var keyByteArray = TextEncodings.Base64Url.Decode(symmetricKeyAsBase64);
        var keyByteArray = Convert.FromBase64String(symmetricKeyAsBase64);

        var signingKey = new HmacSigningCredentials(keyByteArray);

        var issued = data.Properties.IssuedUtc;

        var expires = data.Properties.ExpiresUtc;

        var token = new JwtSecurityToken(_issuer, audienceId, data.Identity.Claims, issued.Value.UtcDateTime,
            expires.Value.UtcDateTime, signingKey);

        var handler = new JwtSecurityTokenHandler();

        var jwt = handler.WriteToken(token);

        return jwt;
    }

    public AuthenticationTicket Unprotect(string protectedText)
    {
        throw new NotImplementedException();
    }
}

```

Here is my Startup class:

```

public void Configuration(IApplicationBuilder app)
{
    HttpConfiguration config = new HttpConfiguration();

    ConfigureOAuthTokenConsumption(app);
    ConfigureOAuthTokenGeneration(app);

    //I tried changing the above order

    //WebApiConfig.Register(httpConfig);

    ConfigureWebApi(config);

    app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
    app.UseWebApi(config);
}

```

```
UnityConfig.RegisterComponents(config);
```

```
TeamManagement.WebApi.App_Start.AutoMapperConfig.RegisterMaps();  
}
```

Can you please help where am doing mistake?

Reply



Senthil says

October 23, 2015 at 7:57 pm

Finally found!!!

```
private void ConfigureOAuthTokenGeneration(IApplicationBuilder app)  
{  
  
    app.CreatePerOwinContext(ApplicationDbContext.Create);  
    app.CreatePerOwinContext(ApplicationUserManager.Create);  
  
    OAuthAuthorizationServerOptions oAuthServerOptions = new OAuthAuthorizationServerOptions()  
    {  
        AllowInsecureHttp = true,  
        TokenEndpointPath = new Microsoft.Owin.PathString("/oauth/token"),  
        AccessTokenExpireTimeSpan = TimeSpan.FromDays(6),  
        Provider = new CustomOAuthProvider(),  
        AccessTokenFormat = new CustomJwtFormat("http://localhost:50658")  
    };  
  
    //app.UseOAuthBearerTokens(oAuthServerOptions);  
    app.UseOAuthAuthorizationServer(oAuthServerOptions);  
}
```

The commented line was the problem. Sorry, it was my fault but that made me to learn more!!!

Thanks a bunch 😊

Reply



Taiseer Joudeh says

October 26, 2015 at 10:41 am

You are welcome, glad you find the issue.

Reply



Senthil says

October 23, 2015 at 8:14 pm

We register allowed Audiences on Startup in both AS & RS.

When client ids are stored in DB, how do RS & AS know about newly registered client ids? – Do we need to restart the app just to get latest added audiences? Is there any better way to do?

I just think, but don't know good or bad:



Can we use ConcurrentList along with Audience DB, I mean when new Audience added, we should add in both Audience store and ConcurrentList and when IIS refresh then on Startup class, all exists Audiences in ConcurrentList again.

Am not sure – it just an idea...

[Reply](#)



**Denys Olleik** says  
October 29, 2015 at 7:35 am

Taiseer, what a great blog, bravo! It helped me a lot coming into world of web development. One suggestion...

I recently stumbled upon trying to figure out if I want to implement refresh token and carry the burden of extra complexity. I needed a way to easily disable the account to prevent user from using their token from making further requests against the api.

I have modified ConfigureOAuthTokenConsumption(). Setting JwtBearerAuthenticationOptions.Provider to a custom provider, you can override OAuthBearerAuthenticationProvider.ValidateIdentity. You can at this point use the user GUID from incoming token to load the actual identity from database and pass it to context.Validated(identity), this gives you ability to always load fresh credentials with every request.

You can add a flag to check if user is active (requires extra column in database to keep track of active users) this way.

[Reply](#)



**Taiseer Joudeh** says  
November 6, 2015 at 4:04 am

Hi Denyes,

The draw back of this approach is hitting the database with each request to validate user identity, well.. this somehow breaks the stateless nature of the bearer tokens, but if you are fine with this you can do it for sure.

[Reply](#)



**Manuel Espino** says  
November 6, 2015 at 6:45 pm

Hi Taiseer

First of all, thank you for this articles. They have helped me to understand OAuth Authentication.

I followed the steps and I don't see errors when I build the project also runs without exceptions, but when I test and I try to obtain the JWT Token, PostMan shows me an error:

“Invalid object name ‘IdentityUserClaims’

This is throwing when the following line is executed:

```
User user = await userManager.FindAsync(context.UserName, context.Password);
```

In my BD already have the table AspNetUserClaim but I've not be able to find out the root of the problem.

Could you please advise me or point me to the right direction?

Thanks in advance.

[Reply](#)



Taiseer Joudeh says

November 11, 2015 at 6:54 pm

You are welcome Manuel.

Did you try to create your own UserStore and RoleStore, or did you try to configure Asp.Net identity database tables?

[Reply](#)



Manuel Espino says

December 16, 2015 at 8:08 am

Sorry for the late response, and yes, the problem was a misspelling in the table names (dang!)

[Reply](#)



Sophonias Dechasa says

November 10, 2015 at 11:50 pm

Hey Taiseer, I just wanted to thank you for your thorough tutorials after combining many of your tutorials I finally created my own middleware. Your tutorials are by far the best I've seen. Keep up the good work. Thanks.

[Reply](#)



Taiseer Joudeh says

November 11, 2015 at 6:44 pm

Always happy to help, thanks for your comment 😊

[Reply](#)



john says

November 11, 2015 at 7:19 am

Hi Taiseer, This has been a great tutorial, I have my own angular front end in development against it. Unfortunately tonight the /oauth/token path has started giving me a 404, I've rolled back several commits to see if I bonked something along the way and no luck.

Is there a way I can verify that the /oauth/token route is being created/bound correctly? Do you have any other suggestions or things I can check? Thank you in advance.

[Reply](#)



Taiseer Joudeh says

November 11, 2015 at 6:44 pm

Hi John,

Giving 404 suddenly means there is change happened on your Angular UI project, I recommend you to use postman to test the back-end and then you can fire fiddler and monitor the requests sent to the Api and see if the URL is correct.

Reply



Reza says

November 12, 2015 at 7:00 am

Hi Taiseer,

Thanks for the article. It has helped me a lot on using oauth and JWT. The solution works flawlessly in IIS express, but deploying to IIS 8.5 I get 404 error on `http://localhost/oauth/token`. Searching for answer, I've updated web.config with all the settings that I found in replies on Stackoverflow. could you please reflect how you host this solution in IIS?

Thanks,  
Reza

Reply



Taiseer Joudeh says

November 16, 2015 at 7:29 pm

Hi Reza, that is strange, currently I'm hosting on Azure Web Apps, I assume they are using latest IIS version.

Reply



Reza says

November 27, 2015 at 6:26 pm

Hi Taiseer,

Thanks for the reply. It was my silliness which I found I had ensured in release mode build, `AllowInsecureHttp` was set to false.

Thanks,  
Reza



Sam says

November 16, 2015 at 8:11 am

Hi Taiseer,

Thank you for uploading above tutorial, Can you please tell me how can I add api version in token signature. I cant

figure out this. Thank you very much again learned a lot from this article.

```
{
  access token:
  token Type:
  expires in:
  Version:
}
```

Regards,  
Sam.

[Reply](#)



Taiseer Joudeh says

November 16, 2015 at 7:20 pm

Hi Sam, if you want to return something in the response of the token endpoint then you can check my SO answer [here](#).

[Reply](#)



Sam says

November 17, 2015 at 7:51 am

Yesterday I found the same but didn't notice it also from you. Thank you much sharing your knowledge.

[Reply](#)



Spike Pierson says

November 17, 2015 at 8:05 pm

Hi Taiseer,

I am posting this in the blog for Part 3 because this is where I'm getting stuck. At the end of Step 4, you direct us to test the ability to start issuing JWT access tokens, by issuing an HTTP POST request to <http://localhost:nnnnn/oauth/token> with the x-www-form-urlencoded data shown in your diagram. I am using Fiddler as my proxy. The POST request looks like this:

```
POST http://localhost:55383/oauth/token HTTP/1.1
User-Agent: Fiddler
Host: localhost:55383
Content-Type: application/x-www-form-urlencoded
Content-Length: 67
```

```
grant_type=password&username=SuperPowerUser&password=MySuperP%40ss!
```

I am consistently getting this response:

```
HTTP/1.1 400 Bad Request
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 87
Content-Type: application/json; charset=UTF-8
Expires: -1
Server: Microsoft-IIS/8.0
Access-Control-Allow-Origin: *
```

```
X-SourceFiles: =?UTF-8?B?
QzpcVXNlcnNcc3BpZXJzb24uQUdFUIBPSU5UXERvY3VtZW50c1xXZWlgaQVBJXEFzcE5ldElkZW50aXR5XEFzcE5ldElkZW50aXR5LldlYkFwaVxvYXV0aF0b2t1bg==?
=
X-Powered-By: ASP.NET
Date: Tue, 17 Nov 2015 16:28:34 GMT
```

```
{"error": "invalid_grant", "error_description": "The user name or password is incorrect."}
```

I found that the reason for this response is because in the CustomOAuthProvider class that was added in Step 1, in the GrantResourceOwnerCredentials method, the call to userManager.FindAsync always returns a null user object. Therefore, the call to context.SetError is always called, and the 400 response is always issued. I tried single-stepping into the userManager.FindAsync method, but to no avail. I know where it's breaking, but have no idea as to why. Is there something that I'm overlooking here? Could you please advise me or point me in the right direction? I would greatly appreciate it. Thanks in advance.

[Reply](#)



Taiseer Joudeh says

November 24, 2015 at 1:45 pm

Hi,

It is basically not matching username/password from the database with what you are trying to send, you can hard-code username/password, test and you should get an access token. So your issue with validating username/password.

Make sure that password is encoded/decoded correctly before sending it to the password manager.

[Reply](#)



Jonathan says

November 18, 2015 at 10:23 am

```
{
  "access_token": "JWT_Token",
  "token_type": "bearer",
  "expires_in": 1799,
  "refresh_token": "8369ac755bdc4e3fa9d3f5870b3af2a0",
  "as:client_id": "10b7c34b2c184d86aa716ecd7a82acb4",
  "userName": "testUser",
  "issued": "Wed, 18 Nov 2015 06:23:04 GMT",
  "expires": "Wed, 18 Nov 2015 06:53:04 GMT"
}
```

Can I replace with the following Json result:

```
{
  "returnCode": 200,
  "message": "Custom Message",
  "data": {
    "access_token": "JWT_Token",
    "token_type": "bearer",
    "expires_in": 1799,
    "refresh_token": "8369ac755bdc4e3fa9d3f5870b3af2a0",
    "as:client_id": "10b7c34b2c184d86aa716ecd7a82acb4",
    "userName": "testUser",
    "issued": "Wed, 18 Nov 2015 06:23:04 GMT",
    "expires": "Wed, 18 Nov 2015 06:53:04 GMT"
  }
}
```

[Reply](#)



Taiseer Joudeh says

November 24, 2015 at 1:38 pm

If you want to change the response you can do this, its up to you

[Reply](#)



Jonathan says

November 30, 2015 at 7:18 am

I mean there is no easy way to change response json format, unless intercept and modify OWIN Body Response Stream. Right?

[Reply](#)



Shaun says

December 16, 2015 at 11:19 pm

Thank you for this tutorial, I now have a working security model for my WEB API, I am creating.

One question though: If IIS is reset, will all outstanding JWTs still work or will they need to be reissued?

[Reply](#)



Taiseer Joudeh says

December 23, 2015 at 6:32 pm

Sure thing, all JWT issues will be working, those are self contained tokens and IIS has nothing to do with them

[Reply](#)



Ronald says

December 21, 2015 at 1:57 pm

Great series! They are really helpfull! But I'm kinda stuck at the moment.

i can get the access\_token from the token endpoint, no problem. but when I try to use the token to reach one of my controller actions with an [Authorize] attribute I get 401.

My get request looks as follows:

url: <http://localhost:51388/api/crm/accounts>

Headers:

Accept: application/json

Content-Type: application/json

Authorization: Bearer



eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1aWQiOiJ3YjYjN2MyYi0wNmZhLTQ4NmEtYjlyZC0wMWJmODRlMjc4YjIiLCJlbmlxdWVfbmFtZSI6IkFkbWluliwiaHR0cD4kis-UAY9RU-KBs

and my controller looks like this:

```
[HttpGet]
[Route("")]
[Authorize]
public IHttpActionResult Accounts()
{
    var accounts = _uow.Repository().GetAll(a => a.ParentAccountId == null, "Addresses", "PhoneNumbers",
    "EmailAddresses");
    var accountDtos = Mapper.Map<IEnumerable, List>(accounts);
    return Ok(accountDtos);
}
```

I don't really know where to start looking to find out why it keeps giving me te 401. I have compared the example code and my code a couple of times, but it looks good te me. Any ideas on where to look?

### Reply



Taiseer Joudah says  
December 24, 2015 at 2:59 am

Please check [this comment](#) which might help in debugging your issue

### Reply



treetopvt says  
December 22, 2015 at 2:00 am

I first want to thank Taiseer for this excellent set of tutorials. I've walked through this and the one on creating a Token server and I've found both to be well-written and a great primer into owin middleware and tokens. If you are having problems with 401s, you can read the story, or jump to my solution!

The first time I walked through this set of blogs I was up and running and moved on to other tasks. I then tried to implement it from memory and was not able to validate endpoints with tokens (401). After going line by line through my code and nuget versions I did not find a difference from my working example.

So, I created another one that was far simpler, removing everything not directly related to webapi2, owin, or tokens, starting from an empty web project. The result, still not able to authenticate tokens with the JwtBearerAuthentication middleware.

This lead me down the rabbit hole of owin middlewares, how they function, are built, etc. Nothing I tried, and I everything the internet could suggest worked for me.

I eventually learned how to build my TokenHandler, deriving from JWTSecurityTokenHandler. Overriding that gave me great insight into how OWIN processes and validates tokens.

I discovered that the ValidateToken function was throwing exceptions, which leads to the token to be considered not valid. The obvious reasons for an invalid token: issuer, audience, keys not matching, were all triple-checked and confirmed to be the same on both ends. So why and where was an exception being raised?

The culprit ended up being the ValidateAudience method. First, I was surprised this was a method and not a function, but I am sure there is a valid reason for this! I do not know why it was failing, but I ended up calling the base method from my overridden ValidateToken function with the code that follows. Result? Success! So two days of banging my head against the OWIN wall I am ready to move on again, with a much greater appreciation/frustration for MS's implementation of middleware.

```
public override ClaimsPrincipal ValidateToken(string securityToken, TokenValidationParameters
```

```
validationParameters, out SecurityToken validatedToken)
{
    var jwt = this.ValidateSignature(securityToken, validationParameters);
    if (validationParameters.ValidateAudience)
    {
        if (validationParameters.AudienceValidator != null)
        {
            if (!validationParameters.AudienceValidator(jwt.Audiences, jwt, validationParameters))
            {
                throw new SecurityTokenInvalidAudienceException(string.Format(CultureInfo.InvariantCulture,
                    ErrorMessages.IDX10231, jwt.ToString()));
            }
        }
        else
        {
            base.ValidateAudience(validationParameters.ValidAudiences, jwt, validationParameters);
        }
    }

    string issuer = jwt.Issuer;
    if (validationParameters.ValidateIssuer)
    {
        if (validationParameters.IssuerValidator != null)
        {
            issuer = validationParameters.IssuerValidator(issuer, jwt, validationParameters);
        }
        else
        {
            issuer = ValidateIssuer(issuer, jwt, validationParameters);
        }
    }

    if (validationParameters.ValidateActor && !string.IsNullOrEmpty(jwt.Actor))
    {
        SecurityToken actor = null;
        ValidateToken(jwt.Actor, validationParameters, out actor);
    }

    ClaimsIdentity identity = this.CreateClaimsIdentity(jwt, issuer, validationParameters);
    if (validationParameters.SaveSigninToken)
    {
        identity.BootstrapContext = new BootstrapContext(securityToken);
    }

    validatedToken = jwt;
    return new ClaimsPrincipal(identity);
}
```

[Reply](#)



**Taiseer Joudah** says

**December 24, 2015 at 12:51 am**

Thanks for sharing this, really appreciated, I never digged too deep in the implementation of the JWTTokenSecurityHandler. I just replied on your SO comment but things are clear now. Thanks again for sharing this.

[Reply](#)



Bo Stokholm says

December 22, 2015 at 1:07 pm

Hi

Great tutorial. I do just have one problem. When I login I get the token but when I try to pass the token to changepassword api then I get a 401. It is like it doesn't even try to validate the token?

I have made a stackoverflow ticket on it: <https://stackoverflow.com/questions/34405490/web-api-giving-401-on-jwt>

Hope you can help.

Reply



Taiseer Joudeh says

December 23, 2015 at 6:29 pm

Hi,

Is this happening only for this endpoint or you are receiving 401 on other endpoints attribute with [Authorize] attribute?

Reply



treetopvt says

December 23, 2015 at 6:58 pm

I've posted a potential solution over on StackOverflow and otherplaces on this Blog, but you basically want to create your own JWTTokenHandler that derives from JWTSecurityTokenHandler. This will at least let you 'open the hood' and see why your token is failing Authorization.

Reply



Felix Rabinovich says

January 24, 2016 at 11:45 pm

I think I figured out what the problem is... issued should be in local time and not in UTC. I put a longer reply with code snippets in the same StackOverflow post.

Reply



Christopher McCrum says

December 24, 2015 at 6:56 am

Hey Taiseer, I was wondering if you'll be offering a write up for ASP NET 5? Your walkthrough has worked flawlessly for us in our older project. I'm currently trying to implement it but I've notice quite a bit has changed. thanks again for all your hard work!

Reply

**Taiseer Joudeh** says

December 24, 2015 at 10:55 pm

Hi Chris,

I'm just waiting for RC 2 as the ASP.NET team keeps breaking some methods which each release, as well the Identity and token issue still not clear in ASP.NET 5 yet, there is no built in middleware to issue access token in ASP.NET 5. Maybe I will use the external IdSrv3 or Open Id connection middleware to do this task.

[Reply](#)**alexstrakh** says

January 11, 2016 at 10:51 am

Excellent explanation and it works fine. One issue here is that when the token is invalid or not specified I'm getting 404 (NotFound) instead of 401 (Unauthorized). I configured my OAuth exactly like you described here. What could be wrong?

[Reply](#)**Taiseer Joudeh** says

January 17, 2016 at 11:48 am

Hi Alex, sorry for the late reply.

Did you find solution for your issue? I never faced such issue when the API returns 404 when it is unauthorized.

[Reply](#)**alexstrakh** says

January 17, 2016 at 11:06 pm

I have noticed that if I also getting redirect to Login.aspx even though I disabled forms auth in web.config.

It seems that after publishing to azure forms auth module continue to work. I found similar issue here and as one of solution is to remove that module:

<http://haacked.com/archive/2011/10/04/prevent-forms-authentication-login-page-redirect-when-you-donrsquot-want.aspx/>

[Reply](#)**Leo** says

February 2, 2016 at 11:01 pm

Hey Taiseer, I am making a call web api in order to authenticate users from asp.net application

```
private static async Task GetAPIToken(string userName, string password, string apiBaseUri)
{
    using (var client = new HttpClient())
    {
```

```
//setup client
client.BaseAddress = new Uri(apiBaseUri);
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

//setup login data
var formContent = new FormUrlEncodedContent(new[]
{
    new KeyValuePair("grant_type", "password"),
    new KeyValuePair("username", userName),
    new KeyValuePair("password", password),
});

//send request
HttpResponseMessage responseMessage = await client.PostAsync("/Token", formContent);

//get access token from response body
var responseJson = await responseMessage.Content.ReadAsStringAsync();
var jobject = JObject.Parse(responseJson);
return jobject.GetValue("access_token").ToString();
}
}
```

The PostAsync is executed successfully on the web api but I could not get the response message in my web application.

Any ideas ?

Thanks in advance,

Leo

[Reply](#)



**Seng Joon** says

February 14, 2016 at 6:03 pm

Help Need. The system works well with create user, delete user, list all the users. however, it fail to obtain token. even i supplied the right username and password, it continue give me error message invalid\_grant. I use debugger to trace, and found ApplicationUser user = await userManager.FindAsync(context.UserName, context.Password); return a Null value.

Exception:Caught: "Could not load file or assembly 'Microsoft.VisualStudio.Web.PageInspector.Runtime, Version=14.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The system cannot find the file specified." (System.IO.FileNotFoundException)

A System.IO.FileNotFoundException was caught: "Could not load file or assembly 'Microsoft.VisualStudio.Web.PageInspector.Runtime, Version=14.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The system cannot find the file specified."  
Time: 14/02/2016 10:50:51 PM  
Thread:[28136]

Any Idea what could be wrong?

[Reply](#)



**Taiseer Joudah** says

February 16, 2016 at 6:35 am

Maybe [this answer](#) will help you

[Reply](#)



shengqisong says

February 18, 2016 at 9:40 pm

This is the best tutorial that I can found. I have a error in the last step, it says: JwtSecurityTokenHandler.cs not found. Do I need to create this class? Thanks for your help.

Reply



Taiseer Joudeh says

February 22, 2016 at 12:29 pm

Download the source code from GitHub and check if this class is created or from NuGet package.

Reply



napster0908 says

February 20, 2016 at 3:43 pm

Hi,

I tried your solution on azure and it works perfectly. However when I try your solution on another server I get the notification below. It only happens on the first login attempt. The second, right after the first, works perfectly. Any idea on what could be causing this?

XMLHttpRequest cannot load <https://api-address/oauth/token>. Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'https://client-address' is therefore not allowed access. The response had HTTP status code 400.

Reply



Taiseer Joudeh says

February 22, 2016 at 12:16 pm

I'm not sure why this is happening, but I recall I faced the same issue and we discussed it on the comments section for this post. Can you please check [this answer](#)

Reply



napster0908 says

February 22, 2016 at 12:38 pm

This is exactly the solution which helped me. Unfortunately I found it out myself yesterday. After I placed the usecors on top an error in my application log started to arise regarding duplicate 'Access-Control-Allow-Origin'. After removing the line in the article you send it disappeared. Thanks for your reply though 😊

Reply





Amol Kulkarni says

February 22, 2016 at 10:43 am

Hi

This is an excellent article.

here is one issue I am facing , in the GrantResourceOwnerCredentials if the user is not found either because of wrong username or password , A set error method is called

```
if (user == null)
{
    context.SetError("invalid_grant", "The user name or password is incorrect.");
    return;
}
```

In the above situation the HttpStatus code is always set to 400 (bad request) . I have tried various approaches but could not set the HttpStatus code to 401. Do you know if there is a way to do it ? I have been struggling with this for some time now.

[Reply](#)



Taiseer Joudeh says

February 22, 2016 at 11:40 am

Hi Amol,

Returning 400 is the correct http status in this case, and this based OAuth 2.0 specs, the user is not authenticated yet and he is providing incorrect username/password so he is providing incorrect data from the client, so 400 is correct. You should return 401 if the user was already authenticated (has an access token) and this token is expired or invalid anymore.

[Reply](#)



Ole H says

February 28, 2016 at 9:37 pm

Hi.

Thank you for these posts, they make a very difficult subject comprehensible.

However I am still a bit confused in what the difference is between the tutorial here on this page, where we are issuing

JWT tokens with oauth insted of "default access tokens" and the tutorial in this 5 part series where we are issuing "OAuth Bearer Tokens".

<http://bitoftech.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/>

As I understand both these tokens are fully self-contained, and there is no corresponding session on the server for any of them, so what is actually the practical difference between issuing and using these two type of tokens?

Is it just that you can decouple Authorization server from Resource server with JWT?

Thanks!

[Reply](#)



Taiseer Joudeh says

March 3, 2016 at 11:56 pm

Hi Ole,

The tutorial [here](#) is using Bearer tokens as the [post here](#) too, the difference only relies on the access token format, the JWT tokens are signed only (not encrypted) and you can decode the claims in the client side application, you can use JWT.io to decode the JWT and see the claims, on the other side, the JWT format is becoming the more standard way of sending OAuth 2.0 bearer tokens.

Hope this answers your question.

[Reply](#)



Simon says

March 15, 2016 at 2:30 pm

I echo all the thanks, these tutorials are a little bit vital !

I (too) am mystified when you say the Angular front end is a "trusted client". How is it trusted? How is it even identified (authenticated)?

[Reply](#)



Ole H says

March 16, 2016 at 2:56 pm

Thank you for your reply. I just have one more questions.

I want to add login and registration with external provider (facebook, google) to the application.

Can I use the guide you have provided here (<http://bitoftech.net/2014/08/11/asp-net-web-api-2-external-logins-social-logins-facebook-google-angularjs-app/>) as-is, or do I have to make some changes to that code now that I have configured the app to use JWT ?

Thanks.

[Reply](#)



Ole H says

March 22, 2016 at 6:26 pm

To answer my own question: The guide on implementing facebook/google login can be used pretty much as-is if you have followed the guide on this page to issue and consume json web tokens, but some small changes was required. Here is what I changed:

In the first guide you set OAuthBearerAuthenticationOptions as a static member of Owin Startup class and later call

app.UseOAuthBearerAuthentication(OAuthBearerOptions) as shown in the code here:

<https://github.com/tjoudeh/AngularJSAuthentication/blob/master/AngularJSAuthentication.API/Startup.cs#l54>

Skip that part. Instead you set OAuthAuthorizationServerOptions as a static member of Startup class, then you just follow the guide on this page (initialize it and call app.UseOAuthAuthorizationServer(OAuthServerOptions) ).

You also have to call

"app.UseExternalSignInCookie(Microsoft.AspNet.Identity.DefaultAuthenticationTypes.ExternalCookie);" as

explained in the first guide.

<https://github.com/tjoudenh/AngularJSAuthentication/blob/master/AngularJSAuthentication.API/Startup.cs#l40>

Then you can generate a similar JWT as in path "http://localhost:59822/oauth/token" in method GenerateLocalAccessTokenResponse in AccountController.

here is my code:

At the end In method "RegisterExternal" and "ObtainLocalAccessToken":

```
[...]
ClaimsIdentity oAuthIdentity = await user.GenerateUserIdentityAsync(UserManager, "JWT");
var accessTokenResponse = GenerateLocalAccessTokenResponse(oAuthIdentity);

return Ok(accessTokenResponse);

private JObject GenerateLocalAccessTokenResponse(ClaimsIdentity oAuthIdentity) {

    TimeSpan tokenExpiration =
    TimeSpan.FromMinutes(Convert.ToDouble(ConfigurationManager.AppSettings["JWTExpirationTimeInMinutes"]));

    var props = new AuthenticationProperties() {
        IssuedUtc = DateTime.UtcNow,
        ExpiresUtc = DateTime.UtcNow.Add(tokenExpiration),
    };

    var ticket = new AuthenticationTicket(oAuthIdentity, props);

    var accessToken = Startup.OAuthServerOptions.AccessTokenFormat.Protect(ticket);

    JObject tokenResponse = new JObject(
        new JProperty("access_token", accessToken),
        new JProperty("expires_in", tokenExpiration.TotalSeconds.ToString()),
        new JProperty("token_type", "bearer"));

    return tokenResponse;
}
```

Thanks again to Taiseer Joudeh for these guides!

[Reply](#)



**Taiseer Joudeh** says

March 24, 2016 at 12:05 pm

You are most welcome, and thanks for sharing the answer and your thoughts.

[Reply](#)

[« Older Comments](#)

## Leave a Reply

Enter your comment here...

ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5

ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4

Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3

ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration – Part 2

Interview with John about establishing a successful blog

AJAX AngularJS API API Versioning

ASP.NET Attribute Routing Authentication

Autherization Server basic authentication C# CacheCow

Client Side Templating Code First Dependency Injection

Entity Framework ETag Foursquare API

HTTP Caching HTTP Verbs IMDB API IoC Javascript jQuery JSON

JSON Web Tokens JWT Model Factory Ninject OAuth

OData Pagination Resources Association Resource Server

REST RESTful Single Page

Applications SPA Token Authentication

Tutorial Web API Web API 2 Web

API Security Web Service [wordpress.com](#)

[wordpress.org](#)



SEARCH

Search this website...