Database First Approach

This article is about OAuth 2.0 authorization scheme integration with ASP.NET MVC REST Web API.

Asma Khalid May 13 2018

17 46 118.4k

WebApiOAuth2.zip

REST Web API is a light-weight essential component of web development in order to share the data across multiple client machines or devices e.g. mobile devices, desktop applications or any website. Authorization of REST Web API is an equally important part for sharing data across multiple client machines and devices in order to protect data sensitivity from any outside breaches and to authenticate the utilization of the target REST Web API.

Authorization of REST Web API can be done via a specific username/password with the combination of a secret key, but, for this type of authorization scheme, REST Web API access needs to be authenticated per call to the hosting server. Also, we as the owner of the server have no way to verify who is utilizing our REST Web API, whether it's the clients that we have allowed access to or if some malicious user is also using our API(s) without our knowledge. Finally, since the username/password is packed to a base64 format automatically by the browser, if any malicious user traces my browser activity and gets ahold of my REST Web API calls they can easily decrypt base64 format and could use my REST Web API for malicious activities.

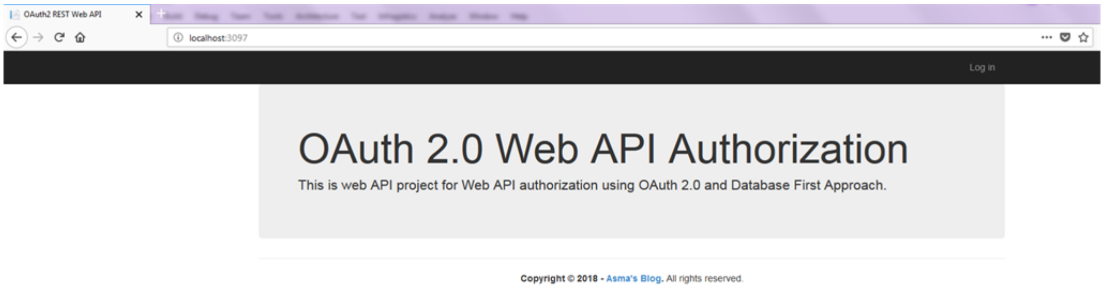
Hmmmmm.....scary stuff! Not to mention that despite the fact that I have authorized my REST Web API, it is still open for malicious users to utilize without even my knowledge. So, what to do? To answer that a new authorization scheme is introduced which can also be utilized in Login flow of any web application as well, but, I will be focusing on it from a REST Web API perspective. So, this new scheme of authorization is OAuth 2.0 which is a token based authorization scheme.

Let's compare OAuth 2.0 authorization scheme to the traditional username/password authorization scheme from REST Web API perspective, i.e.,

	Username/Password REST Web API Authorization	VS	OAuth 2.0 REST Web API Authorization
1.	Access Authorization to same or different REST Web API(s) on the same server is authenticated on each call with provided username/password.		Access Authorization is authenticated only once based on the system's existing user credential, then on successful authorization, an access token is provided for a specific period of time REST Web API(s) call can utilize the access to

2.	REST Web API(s) ... l cannot track the user who is consuming the REST Web API(s). Its utilization is based on mutual trust between Producer and consumer.	Or ... ar Web API(s), so, REST Web API call can track the user who is consuming the REST Web API.
	hackers can easily decrypt the request headers.	access to request header.
4.	All client machines or devices code needs to be updated in case of the change in username/password for malicious activities.	Access token is activated for a specific time period. Change in system user's credential will not require the change of code in target consumer client machines and devices. Update credential generated a new access token.
5.	Username/Password is fixed.	Access token is generated automatically based on system user's credential.

Today, I shall demonstrate OAuth 2.0 mechanism to authorize a REST Web API which will also give us the benefit of [Authorize] attribute via OWIN security layer.



OAuth 2.0 Web API Authorization

This is web API project for Web API authorization using OAuth 2.0 and Database First Approach.

Copyright © 2018 - Asma's Blog. All rights reserved.

Following are a few prerequisites before you proceed any further,

1. Knowledge of OAuth 2.0.
2. Knowledge of ASP.NET MVC5.
3. Knowledge of C# programming.
4. Knowledge of REST Web API.

You can download the complete source code or you can follow the step by step discussion below. The sample code is developed in Microsoft Visual Studio 2015 Enterprise.

Let's begin now:

Step 1

Create a new Web API project and name it "WebApiOauth2".

Step 2

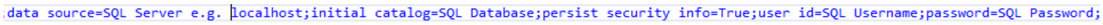
Install the following NuGet packages into your project, i.e.,

1. Microsoft.Owin.Security.OAuth
2. Microsoft.Owin.Cors
3. Microsoft.AspNet.WebApi.Core
4. Microsoft.AspNet.WebApi.Owin

reate "DB_Oauth_ l" database into your SQL Server and execut

```sql
USE [DB_Oauth_API]
GO

DROP PROCEDURE [dbo].[LoginByUsernamePassword]
GO
/****** Object:  Table [dbo].[Login]    Script Date: 5/10/2018 2:37:02 PM ******/
DROP TABLE [dbo].[Login]
GO
/****** Object:  Table [dbo].[Login]    Script Date: 5/10/2018 2:37:02 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Login](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [username] [varchar](50) NOT NULL,
    [password] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Login] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_L
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
SET IDENTITY_INSERT [dbo].[Login] ON

INSERT [dbo].[Login] ([id], [username], [password]) VALUES (1, N'admin', N'adminpass')
SET IDENTITY_INSERT [dbo].[Login] OFF
/****** Object:  StoredProcedure [dbo].[LoginByUsernamePassword]    Script Date: 5/10/2018 2:37:02 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Asma Khalid>
-- Create date: <Create Date,,15-Mar-2016>
-- Description: <Description,,You are Allow to Distribute this Code>
-- =============================================
CREATE PROCEDURE [dbo].[LoginByUsernamePassword]
    @username varchar(50),
    @password varchar(50)
AS
BEGIN
    SELECT id, username, password
    FROM Login
    WHERE username = @username
    AND password = @password
END

GO
```

application. Do also update your SQL server connection string in t ' f downloaded the project i.e.



```
data source=SQL Server e.g. localhost;initial catalog=SQL Database;persist security info=True;user id=SQL Username;password=SQL Password;
```

.

Rename "Controllers/ValueController.cs" file to "Controllers/WebApiController.cs".

Step 5

Open, "Controllers/WebApiController.cs" file replace following code,

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace WebApiOauth2.Controllers
{
    [Authorize]
    public class WebApiController : ApiController
    {
        // GET api/WebApi
        public IEnumerable<string> Get()
        {
            return new string[] { "Hello REST API", "I am Authorized" };
        }

        // GET api/WebApi/5
        public string Get(int id)
        {
            return "Hello Authorized API with ID = " + id;
        }

        // POST api/WebApi
        public void Post([FromBody]string value)
        {
        }

        // PUT api/WebApi/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE api/WebApi/5
        public void Delete(int id)
        {
        }
    }
}
```

In the above code, I have created very simple and basic REST Web API(s). Notice [Authorize] attribute is already placed at the top of the controller to make the REST Web API(s) access secure.

Now open "App_Sta WebApiConfig.cs" file and replace the follow

```
using System;
using System.Collections.Generic;
```

```
using Microsoft.Owin.Security.OAuth;

namespace WebApiOauth2
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services
            // Configure Web API to use only bearer token authentication.
            config.SuppressDefaultHostAuthentication();
            config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.Authenticatio

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            // WebAPI when dealing with JSON & JavaScript!
            // Setup json serialization to serialize classes to camel (std. Json format
            var formatter = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
            formatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Seriali

            // Adding JSON type web api formatting.
            config.Formatters.Clear();
            config.Formatters.Add(formatter);
        }
    }
}
```

In the above code the following two lines of code will add authentication filter for Oauth 2.0 authorization scheme and surpass any existing authorization scheme i.e.

```
surpass any existing authorization scheme i.e.
            // Web API configuration and services
            // Configure Web API to use only bearer token authentication.
            config.SuppressDefaultHostAuthentication();
            config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.Authenticatio
```

Step 7

Now, open "App_Start/Startup.Auth.cs" file and replace following code in it i.e.

```
ng System;
ng Microsoft.AspNet.Identity;
ng Microsoft.AspNet.Identity.Owin;
ng Microsoft.Owin.Security.Google;
ng Owin;
ng WebApiOauth2.Models;
ng Microsoft.Owin.Security.OAuth;
ng WebApiOauth2.Helper_Code.OAuth2;



 public partial class Startup
 {
     #region Public /Protected Properties.

     /// <summary>
     /// OAUTH options property.
     /// </summary>
     public static OAuthAuthorizationServerOptions OAuthOptions { get; private set; }

     /// <summary>
     /// Public client ID property.
     /// </summary>
     public static string PublicClientId { get; private set; }

     #endregion

     // For more information on configuring authentication, please visit http://go.microsoft.co
 Id=301864
     public void ConfigureAuth(IAppBuilder app)
     {
         // Enable the application to use a cookie to store information for the signed in user
         // and to use a cookie to temporarily store information about a user logging in with a
         // Configure the sign in cookie
         app.UseCookieAuthentication(new CookieAuthenticationOptions
         {
             AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
             LoginPath = new PathString("/Account/Login"),
             LogoutPath = new PathString("/Account/LogOff"),
             ExpireTimeSpan = TimeSpan.FromMinutes(5.0),
         });

         app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

         // Configure the application for OAuth based flow
         PublicClientId = "self";
         OAuthOptions = new OAuthAuthorizationServerOptions
         {
             TokenEndpointPath = new PathString("/Token"),
             Provider = new AppOAuthProvider(PublicClientId),
             AuthorizeEndpointPath = new PathString("/Account/ExternalLogin"),
             AccessTokenExpireTimeSpan = TimeSpan.FromHours(4),
             AllowInsecureHttp = true //Don't do this in production ONLY FOR DEVELOPING: ALLOW
         };

         // Enable the application to use bearer tokens to authenticate users
         app.UseOAuthBearerTokens(OAuthOptions);

         // Enables the application to temporarily store user information when they are verifyi
 or authentication process.
         app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes.TwoFactorCookie, TimeSpan.From

         // Enables the application to remember the second login verification factor such as
         // Once you check this option, your second step of verification during the login pr
         //Uncomment the following lines to enable logg
         //app.UseMicrosoftAccountAuthentication(
         //    clientId: "",
         //    clientSecret: "");


         //   consumerKey: "",
         //   consumerSecret: "");

         //app.UseFacebookAuthentication(
         //   appId: "",
         //   appSecret: "");

         //app.UseGoogleAuthentication(new GoogleOAuth2AuthenticationOptions()
         //{
         //    ClientId = "",
         //    ClientSecret = ""
         //});
     }
 }
```

In the above piece of code, "PublicClientId" is used when "AuthorizeEndpointPath" is utilized for unique instantiation from client-side. Following lines of code will enable the OAuth 2.0 authorization scheme, i.e.

```
from client side. Following lines of code will enable the OAuth 2.0 authorization scheme
            // Configure the application for OAuth based flow
            PublicClientId = "self";
            OAuthOptions = new OAuthAuthorizationServerOptions
            {
                TokenEndpointPath = new PathString("/Token"),
                Provider = new AppOAuthProvider(PublicClientId),
                AuthorizeEndpointPath = new PathString("/Account/ExternalLogin"),
                AccessTokenExpireTimeSpan = TimeSpan.FromHours(4),
                AllowInsecureHttp = true //Don't do this in production ONLY FOR DEVELOP
            };

            // Enable the application to use bearer tokens to authenticate users
            app.UseOAuthBearerTokens(OAuthOptions);
```

OAuthAuthorizationOptions are explained as follow i.e.

- *TokenEndpointPath* ->
  This is the path which will be called in order to authorize the user credentials and in return it will return the generated access token.

- *Provider* ->
  You need to implement this class (which I have in this tutorial) where you will verify the user credential and create identity claims in order to return the generated access token.

- *AuthorizeEndpointPath* ->
  In this tutorial, I am not using this property as I am not taking consent of external logins. So, if you are using external logins then you can update this path to get user consent then required access token will be

AccessTokenExpireTimeSpan ->
This is the time period you want your access token to be accessible. The shorter time span is recommended for sensitive API(s).

Use this property for developer environment.

More properties can be studied here.

Step 8

Now, create "Helper_Code/OAuth2/AppOAuthProvider.cs" file and replace following code in it i.e.

```
//-----------------------------------------------------------------------
// <copyright file="AppOAuthProvider.cs" company="None">
//     Copyright (c) Allow to distribute this code.
// </copyright>
// <author>Asma Khalid</author>
//-----------------------------------------------------------------------

namespace WebApiOauth2.Helper_Code.OAuth2
{
    using Microsoft.Owin.Security;
    using Microsoft.Owin.Security.Cookies;
    using Microsoft.Owin.Security.OAuth;
    using Models;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Security.Claims;
    using System.Threading.Tasks;
    using System.Web;

    /// <summary>
    /// Application OAUTH Provider class.
    /// </summary>
    public class AppOAuthProvider : OAuthAuthorizationServerProvider
    {
        #region Private Properties

        /// <summary>
        /// Public client ID property.
        /// </summary>
        private readonly string _publicClientId;

        /// <summary>
        /// Database Store property.
        /// </summary>
        private Oauth_APIEntities databaseManager = new Oauth_APIEntities();

        #endregion

        #region Default Constructor method.

        /// <summary>
        /// Default Constructor method.
        public AppOAuthProvider(string publicClientId)
        {
            //TODO: Pull from configuration
            if (publicClientId == null)
            {
```

```

            // Settings.
            _publicClientId = publicClientId;
        }

        #endregion

        #region Grant resource owner credentials override method.

        /// <summary>
        /// Grant resource owner credentials overload method.
        /// </summary>
        /// <param name="context">Context parameter</param>
        /// <returns>Returns when task is completed</returns>
        public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwner
        {
            // Initialization.
            string usernameVal = context.UserName;
            string passwordVal = context.Password;
            var user = this.databaseManager.LoginByUsernamePassword(usernameVal, passwor

            // Verification.
            if (user == null || user.Count() <= 0)
            {
                // Settings.
                context.SetError("invalid_grant", "The user name or password is incorre

                // Retuen info.
                return;
            }

            // Initialization.
            var claims = new List<Claim>();
            var userInfo = user.FirstOrDefault();

            // Setting
            claims.Add(new Claim(ClaimTypes.Name, userInfo.username));

            // Setting Claim Identities for OAUTH 2 protocol.
            ClaimsIdentity oAuthClaimIdentity = new ClaimsIdentity(claims, OAuthDefault
            ClaimsIdentity cookiesClaimIdentity = new ClaimsIdentity(claims, CookieAuthe

            // Setting user authentication.
            AuthenticationProperties properties = CreateProperties(userInfo.username);
            AuthenticationTicket ticket = new AuthenticationTicket(oAuthClaimIdentity, p

            // Grant access to authorize user.
            context.Validated(ticket);
            context.Request.Context.Authentication.SignIn(cookiesClaimIdentity);
        }

        #endregion

        #region Token endpoint override method.
        /// Token endpoint override method
        /// </summary>
        /// <param name="context">Context parameter</param>
        /// <returns>Returns when task is completed</returns>
        public override Task TokenEndpoint(OAuthTokenEndpointContext context)
```

```
            {
                // Adding.
                context.AdditionalResponseParameters.Add(property.Key, property.Value);
            }

            // Return info.
            return Task.FromResult<object>(null);
        }

        #endregion

        #region Validate Client authntication override method

        /// <summary>
        /// Validate Client authntication override method
        /// </summary>
        /// <param name="context">Contect parameter</param>
        /// <returns>Returns validation of client authentication</returns>
        public override Task ValidateClientAuthentication(OAuthValidateClientAuthenticat
        {
            // Resource owner password credentials does not provide a client ID.
            if (context.ClientId == null)
            {
                // Validate Authoorization.
                context.Validated();
            }

            // Return info.
            return Task.FromResult<object>(null);
        }

        #endregion

        #region Validate client redirect URI override method

        /// <summary>
        /// Validate client redirect URI override method
        /// </summary>
        /// <param name="context">Context parmeter</param>
        /// <returns>Returns validation of client redirect URI</returns>
        public override Task ValidateClientRedirectUri(OAuthValidateClientRedirectUriCo
        {
            // Verification.
            if (context.ClientId == _publicClientId)
            {
                // Initialization.
                Uri expectedRootUri = new Uri(context.Request.Uri, "/");

                // Verification.
                if (expectedRootUri.AbsoluteUri == context.RedirectUri)
                {
                    // Validating.
                    context.Validated();
                }
            // Return info.
            return Task.FromResult<object>(null);
        }

        #endregion
```

```

        /// <summary>
        /// Create Authentication properties method.
        /// </summary>
        /// <param name="userName">User name parameter</param>
        /// <returns>Returns authenticated properties.</returns>
        public static AuthenticationProperties CreateProperties(string userName)
        {
            // Settings.
            IDictionary<string, string> data = new Dictionary<string, string>
                                               {
                                                   { "userName", userName }
                                               };

            // Return info.
            return new AuthenticationProperties(data);
        }

        #endregion
    }
}
```

In the above code, "GrantResourceOwnerCredentials(...)" method is the key method which is called when TokenEndpointPath is called. Notice that "GrantResourceOwnerCredentials(...)" method is used by "grant_type=password" scheme. If you are using "grant_type=client_credentials" scheme then you need to override "GrantClientCredentials(...)" method. Other methods are part of "OAuthAuthorizationServerProvider" class, use them as they are. In "GrantResourceOwnerCredentials(...)" method we are verifying the system login user and then create the require identities claims and then generate the returning access token ticket i.e.

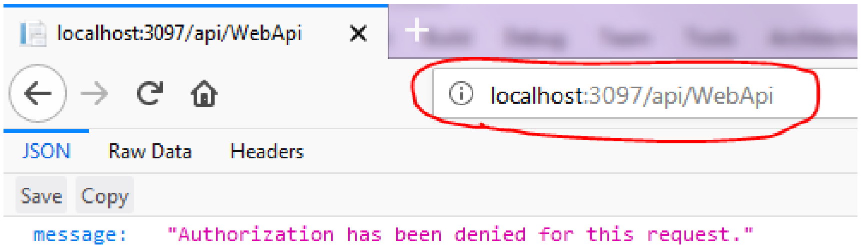
```
#region Grant resource owner credentials override method.

/// <summary>
/// Grant resource owner credentials overload method.
/// </summary>
/// <param name="context">Context parameter</param>
/// <returns>Returns when task is completed</returns>
public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredent
{
    // Initialization.
    string usernameVal = context.UserName;
    string passwordVal = context.Password;
    var user = this.databaseManager.LoginByUsernamePassword(usernameVal, passwordVal).T

    // Verification.
    if (user == null || user.Count() <= 0)
    {
        // Settings.
        context.SetError("invalid_grant", "The user name or password is incorrect.");

        // Retuen info.
        return;
    // Initialization.
    var claims = new List<Claim>();
    var userInfo = user.FirstOrDefault();

    // Setting
```

```
    // Setting Claim Identities for OAUTH 2 protocol.
    ClaimsIdentity oAuthClaimIdentity = new ClaimsIdentity(claims, OAuthDefaults.Authent
    ClaimsIdentity cookiesClaimIdentity = new ClaimsIdentity(claims, CookieAuthenticati

    // Setting user authentication.
    AuthenticationProperties properties = CreateProperties(userInfo.username);
    AuthenticationTicket ticket = new AuthenticationTicket(oAuthClaimIdentity, properti

    // Grant access to authorize user.
    context.Validated(ticket);
    context.Request.Context.Authentication.SignIn(cookiesClaimIdentity);
}

#endregion
```

Step 9

Now, execute the project and use the following link in the browser to see your newly created REST Web API method in action as follows:

```
yourlink:port/api/WebApi
```

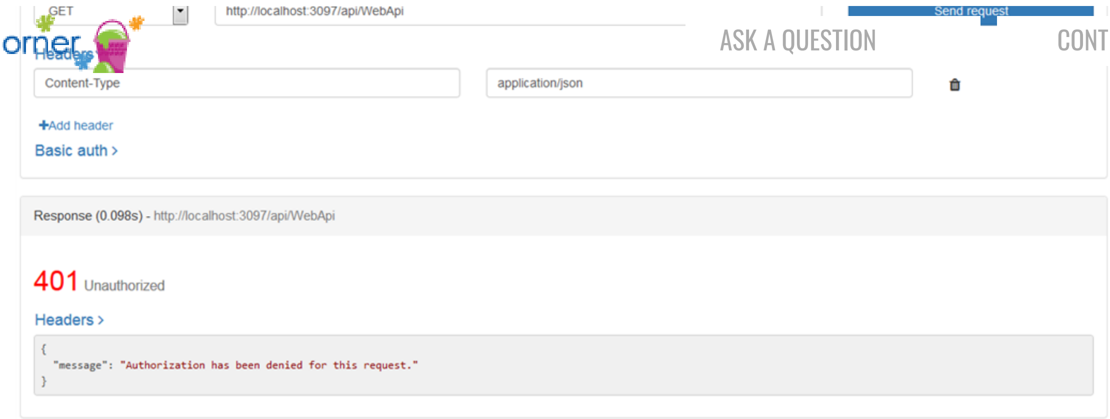


```
message:   "Authorization has been denied for this request."
```

In the above snippet, you will notice that since now our REST Web API has been authorized, therefore, we cannot directly execute the REST Web API URL in the browser.

Step 10

Let's test out REST Web API in REST Web API client. I am using Firefox plugin i.e. "**RESTED**". At, first, I simply try to hit the REST Web API without any authorization details and I will get following response i.e.



GET http://localhost:3097/api/WebApi

Content-Type: application/json

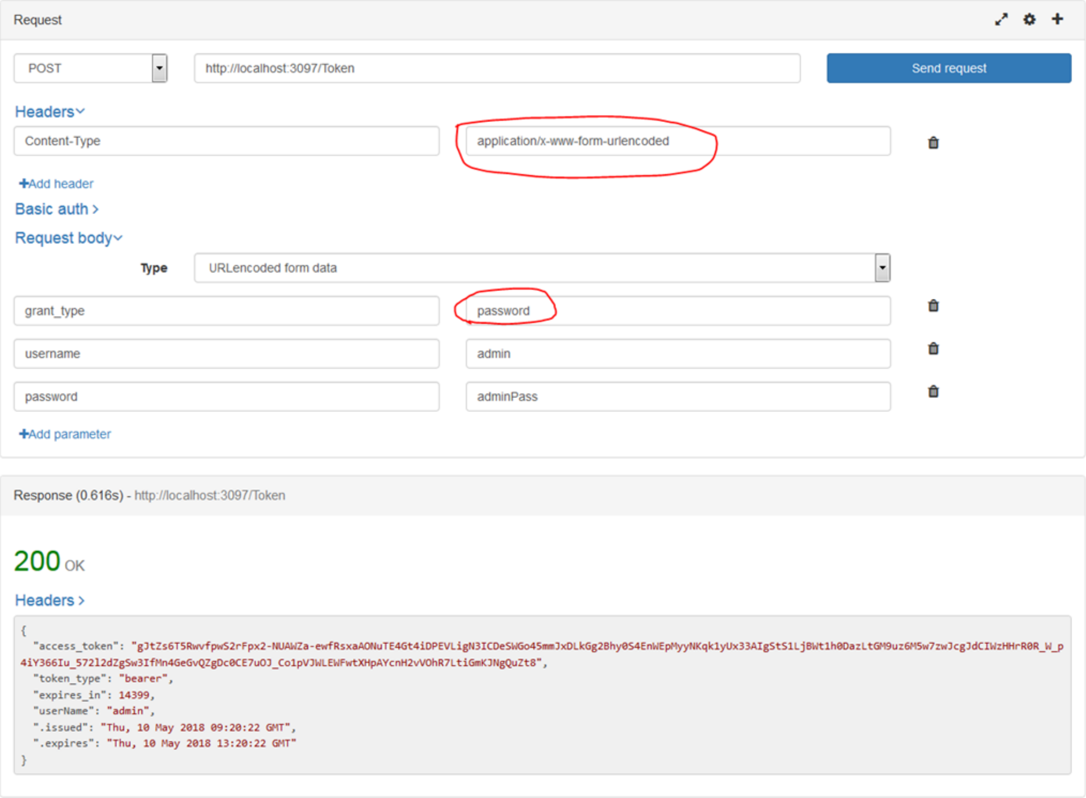
Response (0.098s) - http://localhost:3097/api/WebApi

401 Unauthorized

```
{
  "message": "Authorization has been denied for this request."
}
```

Step 11

Now, I will provide the system user authorization to get access token and then use that access token as a header in the REST Web API and try to his the REST Web API which will return the following response, i.e.



POST http://localhost:3097/Token

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

Type: URLencoded form data

grant_type: password

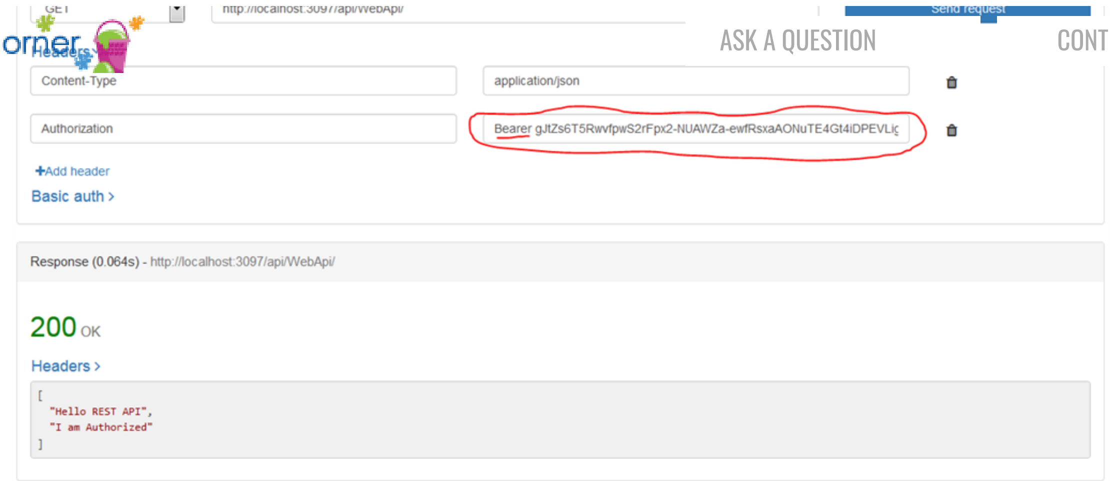
username: admin

password: adminPass

Response (0.616s) - http://localhost:3097/Token

200 OK

```
{
  "access_token": "gJtZs6T5RwvfpwS2rFpx2-NUAWZa-ewfRsxaAONuTE4Gt4iDPEVLigN3ICDeSWGo45mmJxDLkGg2Bhy0S4EnWEpMyyNKqk1yUx33AIgStS1LjBWt1h0DazLtGM9uz6M5w7zwJcgJdCIWzHHrR0R_W_p
4iY366Iu_572l2dZgSw3IfMn4GeGvQZgDc0CE7uOJ_Co1pVJWLEWFwtXHpAYcnH2vVOhR7LtiGmKJNgQuZt8",
  "token_type": "bearer",
  "expires_in": 14399,
  "userName": "admin",
  ".issued": "Thu, 10 May 2018 09:20:22 GMT",
  ".expires": "Thu, 10 May 2018 13:20:22 GMT"
}
```



GET http://localhost:3097/api/WebApi/ Send request

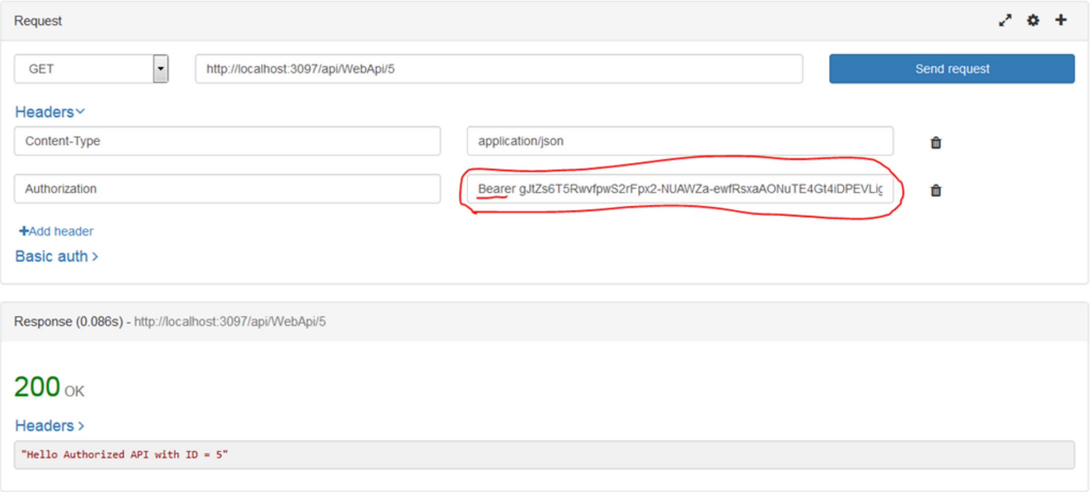
Headers

Content-Type	application/json
Authorization	Bearer gJtZs6T5RwvfpwS2rFpx2-NUAWZa-ewfRsxaAONuTE4Gt4iDPEVLi

Response (0.064s) - http://localhost:3097/api/WebApi/

200 OK

```
[
  "Hello REST API",
  "I am Authorized"
]
```



Request

GET http://localhost:3097/api/WebApi/5 Send request

Headers

Content-Type	application/json
Authorization	Bearer gJtZs6T5RwvfpwS2rFpx2-NUAWZa-ewfRsxaAONuTE4Gt4iDPEVLi

Response (0.086s) - http://localhost:3097/api/WebApi/5

200 OK

```
"Hello Authorized API with ID = 5"
```

Notice in the above snippets that access token is provided as "Authorization" header with "Bearer access_token" scheme in order to call the REST Web API. Also, notice the path when the token is being generated i.e. "{your_site_url}/Token".

Conclusion

In this article, you learned about OAuth 2.0 authorization scheme integration with ASP.NET MVC REST Web API. You also learned about the short comparison between user/password based authorization and OAuth 2.0 token based authorization. You also learned about OAuth 2.0 scheme authentication mechanism for local system users with the Entity Framework database first approach.

cus

C# Corner

chain FAQ

ASK A QUESTION CONTRIBUTE

ASP.NET MVC OAuth 2.0 REST Web API

Asma Khalid TOP 500

Computer Programmer by Profession, Computer Scientist by Heart, Fanatic Explorer, Technology Centric. I am a Versatile Computer Science Evangelist. Enjoy doodling with technology.

http://www.asmak9.com/

190 2.2m 3 2

View Previous Comments

17 46

Type your comment here and press Enter Key (Minimum 10 characters)

Or how would i implement token refresh

Vaughan Trebilco Mar 03, 2019

1710 56 0 0 1 Reply

Look into this thread https://bit.ly/2DnfDLy

Asma Khalid Mar 11, 2019

190 9.6k 2.2m 0

Hello Asma. Is there a way to make the token never expire?

Vaughan Trebilco Mar 03, 2019

1710 56 0 0 1 Reply

Token will eventually expire.

Asma Khalid Mar 11, 2019

190 9.6k 2.2m 0

Hello Asma. In a controller I can get the user name associated with a token like this: User.Identity.Name; How do I get the token value so I can store it in a database?

Vaughan Trebilco Feb 24, 2019

1710 56 0 0 1 Reply

You need to store the access token as claims cookies then you can use following line to retrieve your access token in Controller var authenticateInfo = await HttpContext.Authentication.GetAuthenticateInfoAsync("Bearer");string accessToken = authenticateInfo.Properties.Items[".Token.access_token"];

Hello Asma. sked this question previously and passed your rep yc below? “The delete is for a user to logout of the app. I’m not sure what you read but it’s a very basic functionality of any service I know of. All you have to do on your end is either delete the token or flag it as such so that it can’t be used.”

Vaughan Trebilco Feb 12, 2019

1710 56 0 0 1 Reply

Here is a thing you can not delete the token as it is cached as cookies. They only way that works is to either explicitly/forced log out or flag the token in your DB. The will expire at its expiration time time only

Asma Khalid Feb 13, 2019

190 9.6k 2.2m 0

Hi, thanks for this article, can this generated token be used in asp.net mvc controller to authorize action methods? (not from javascript). Imagine we authenicate asp.net mvc with the api and get the token and save it in a cookie. Then how can we auhtorize action methods by using that token?

reza shirazi Feb 10, 2019

1759 7 0 0 2 Reply

Use [Authorize] annotation at method level instead of controller level.

Asma Khalid Feb 11, 2019

190 9.6k 2.2m 0

Hi Asma, I was asking about a separate MVC project where it authenticate and authorize itslef using the API project. Appreciate your reponse.

reza shirazi Feb 12, 2019

1759 7 0 0

Thanks again this article has been extremely helpful. I’m not sure why but I have a spec that asks to delete the registration despite the timeout feature. Response should be “Registration is removed. The token is now invalid”. Do you know how this would be done?

Vaughan Trebilco Feb 07, 2019

1710 56 0 0 1 Reply

Revoking explicit token access is difficult as it goes against its nature since the whole process works with cookie expiration. But, to achieve it you need to either explicitly sign out the user access the token and do a little work to store token in Db and then do verification via Db and remove the token to revoke access. Look into this thread https://bit.ly/2DnfDLy

Asma Khalid Feb 08, 2019

190 9.6k 2.2m 0

Wow, this was awesome. I searched high and low for something like this: updated, simple, beautiful, full example. It was hard! Kudos to you!

Marco Gaertner Feb 02, 2019

1763 3 0 0 1 Reply

Thank you.

Asma Khalid Feb 04, 2019

190 9.6k 2.2m 0

How i can logout Specific user

Osama Ahmed Dec 31, 2018

1758 8 0 0 1 Reply

Look into this https://bit.ly/2GPJukL

Hi Asma nice tutorial. When i run code in VS it works but when i try to use it in IIS 7 I get folowinfg error:The controller for path "Token" was not found or does not implement IController

Niklas Ryden

Dec 13, 2018

1747 19 0

0 6 Reply 6

[HttpException]: The controller for path '/EAVismaBusinessAPI_deploy/token' was not found or does not implement IController. at System.Web.Mvc.DefaultControllerFactory.GetControllerInstance(RequestContext requestContext, Type controllerType) at System.Web.Mvc.MvcHandler.ProcessRequestInit(HttpContextBase httpContext, IController& controller, IControllerFactory& factory) at System.Web.Mvc.MvcHandler.BeginProcessRequest(HttpContextBase httpContext, AsyncCallback callback, Object state) at System.Web.HttpApplication.CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute() at System.Web.HttpApplication.ExecuteStepImpl(IExecutionStep step) at System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)

Niklas Ryden

Dec 14, 2018

1747 19 0

0

I can get other controllers e.g http://eatest:8081/api/StockBalace with [Authorize] removed on the controller to work

Niklas Ryden

Dec 14, 2018

1747 19 0

0

I have tested the deployment on IIS7 & IIS8 and it is working fine. Tell me at what point this error occurs? How are you consuming the API? Ensure you have deploy the package with Release mode. I have tested in Chrome & Firefox.

Asma Khalid

Dec 14, 2018

190 9.6k 2.2m

0

Yes I know but I need to call other methods that require username as a parameter. So how do I get the username that was used for the initial username and password login?

Vaughan Trebilco

Dec 12, 2018

1710 56 0

0 3 Reply

It depends on your workflow. In case of API user credentials or any secret key is known to consumer of the API with mutual trust. In case of OAuth authorization via login workflow you can store require user info in session to access it globally within the session of particular user. Observe in my provided consumption via REST client that I have provided the user credentials first to acquire the token. This means I already have knowledge of the login credentials. I hope this solve your issue.

Asma Khalid

Dec 12, 2018

190 9.6k 2.2m

0

Hi again and thanks again for you very good example. the answer to my question turned out as easy as this. I just added "string username = User.Identity.Name;" to the get method.

Vaughan Trebilco

Dec 13, 2018

1710 56 0

0

Great to know....

Asma Khalid

Dec 14, 2018

190 9.6k 2.2m

0

cus
C# Corner
chain FAQ

ASK A QUESTION CONTRIBUTE

OUR TRAINING

Mastering Node.js
Learn how to build create mobile and Web apps using Node.js

TRENDING UP

01 C# Coding Standards 😎

02 GraphQL In .NET Core Web API With Entity Framework Core - Part Three

04 ASP.NET MVC Request Life Cycle

05 CRUD Operations On CosmosDB Mongo API Using ASP.NET Core

06 Filtration, Sorting, And Pagination In Angular 7 Using Web API And SQL Server

07 Upload Files In Azure Blob Storage Using ASP.NET Core

08 GraphQL In .NET Core Web API With Entity Framework Core - Part One

09 What Does “var” Mean In C#?

10 What's New In C# 8

View All ○

About Us Contact Us Privacy Policy Terms Media Kit Sitemap Report a Bug FAQ Partners C# Tutorials