Menu Superior

Implementando Bearer Autentication com WebAPI e Owin

Friday, October 3, 2014

webapi (/andrebaltieri/Tags/webapi) | security (/andrebaltieri/Tags/security) | owin (/andrebaltieri/Tags/owin) | oauth (/andrebaltieri/Tags/oauth) | token (/andrebaltieri/Tags/token)

Introdução

Durante praticamente todos projetos que trabalhamos, nos deparamos com a necessidade da implementação de recursos de segurança, como autenticação e autorização. No desenvolvimento de APIs isto não é diferente, pois em vários endpoints teremos o conteúdo restrito a determinados usuários e grupos.

Como funciona a autenticação no WebAPI

Ao contrario de aplicações ASP.NET convencionais, quando trabalhamos com serviços (WebAPI) não temos os dados do usuário em uma sessão, ou seja, você não se autentica uma vez e simplesmente faz as requisições que quiser, você deve se autenticar a cada requisição.

Isto acontece pois o contexto de serviços é diferente das aplicações convencionais, onde temos a duração do mesmo para apenas uma requisição. Resumidamente, a requisição acontece, você autentica seu usuário, retorna o resultado (200, 404, 401) e essa requisição termina, morre!

Existem alguns tipos de autenticação como Bearer e Basic, onde eu particularmente prefiro o Bearer, pois ele trafega um token e não um usuário/senha no header. Embora o usuário/senha trafegados no header da autenticação Basic sejam encriptados em Base64, eles são facilmente desencriptados com o próprio Fiddler.

Como comentei anteriormente, a cada requisição devemos enviar o Token, sendo assim, uma requisição a um endpoint ficaria neste formato:

Bearer

Content-Type: application/json
Authorization: Bearer SEUTOKENAQUI

Basic

Content-Type: application/json
Authorization: Basic USER_PASS_EM_BASE64

Com base nestas informações, temos as seguintes tarefas para a autenticação:

- Obter um Token
- Fazer um request enviando o Token
- Validar o token
- Retornar o resultado da requisição

Neste post não vou entrar em méritos de como você está recuperando seus usuários. Estou partindo do ponto que você tem um repositório de usuários e alguma forma de validar seu usuário e senha.

Projeto e Dependências

Para este projeto, vou utilizar alguns pacotes para geração de Token, que por usa vez já se baseiam no Owin, então iniciamos o projeto com um “Empty Web Application” e marcamos a opção “WebAPI” para trazer somente o necessário para a aplicação.

Os pacotes necessários são:

- Install-Package Microsoft.AspNet.WebApi.Owin
- Install-Package Microsoft.Owin.Host.SystemWeb
- Install-Package Microsoft.Owin.Security.OAuth
- Install-Package Microsoft.Owin.Cors

Startup

Como estamos trabalhando com o Owin, temos tudo concentrado na classe Startup.cs, na raiz da aplicação. Sendo assim, nossa classe Startup.cs fica desta forma:

```
1  using Microsoft.Owin;
2  using Microsoft.Owin.Security.OAuth;
3  using OAuthServer.Api.Security;
4  using Owin;
```

```
using System;
using System.Web.Http;

namespace OAuthServer.Api
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            HttpConfiguration config = new HttpConfiguration();

            ConfigureOAuth(app);

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

        public void ConfigureOAuth(IAppBuilder app)
        {
            OAuthAuthorizationServerOptions OAuthServerOptions = new OAuthAuthorizationServerOptions()
            {
                AllowInsecureHttp = true,
                TokenEndpointPath = new PathString("/token"),
                AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),
                Provider = new SimpleAuthorizationServerProvider()
            };

            // Token Generation
            app.UseOAuthAuthorizationServer(OAuthServerOptions);
            app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());

        }
    }
}
```

oauthstartupcs view raw (https://gist.github.com/andrebaltieri/302f461592c97d8f8c61/raw/36d47ea1b663db72ad616fc4e71ab8c438cb9160/oauthstartupcs) (https://gist.github.com/andrebaltieri/302f461592c97d8f8c61#file-oauthstartupcs) hosted with ♥ by **GitHub (https://github.com)**

Como podemos ver, o único ponto diferente nesta classe é o método ConfigureOAuth, onde vamos cofigurar a segurança.

Para ter um provedor de Tokens na sua API com Owin + Auth, basicamente o que precisamos é uma configuração básica, dada pela instância do OAuthAuthorizationServerOptions, onde informamos itens como o endpoint on de token será gerado (Linha 28), tempo de vida do token (Linha 29) e o ainda não criado Provider (Linha 30).

Os itens mais importantes aqui são o Provider e o TokenEndPoint, pois são através deles que vamos gerar o token de acesso.

Criando um Provider

A criação de um provider nada mais é do que a implementação de uma classe que herde de OAuthAuthorizationServerProvider, cujo internamente terá dois métodos, ValidateClientAuthentication e GrantResourceOwnerCredentials. Destes métodos, utilizaremos o GrantResourceOwnerCredentials para validar um usuário e senha. Deste modo, a classe fica assim:

```
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.Owin.Security.OAuth;
using OAuthServer.Data.Repositories;
using OAuthServer.Domain.Contracts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web;

namespace OAuthServer.Api.Security
{
    public class SimpleAuthorizationServerProvider : OAuthAuthorizationServerProvider
```

```
15      {
16          public override async Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
17          {
18              context.Validated();
19          }
20
21          public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
22          {
23
24              context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "*" });
25              using (IUserRepository _repository = new UserRepository(new Data.DataContexts.OAuthServerDataContext()))
26              {
27                  var user = _repository.Authenticate(context.UserName, context.Password);
28
29                  if (user == null)
30                  {
31                      context.SetError("invalid_grant", "The user name or password is incorrect.");
32                      return;
33                  }
34              }
35
36              var identity = new ClaimsIdentity(context.Options.AuthenticationType);
37              identity.AddClaim(new Claim("sub", context.UserName));
38              identity.AddClaim(new Claim("role", "user"));
39
40              context.Validated(identity);
41          }
42      }
43  }
```

view raw (https://gist.github.com/andrebaltieri/4cd59fe48e3fba7add03/raw/be4745819b27500ea5673b4abe3919e53b585f86/SimpleOauthProvider) SimpleOauthProvider (https://gist.github.com/andrebaltieri/4cd59fe48e3fba7add03#file-simpleoauthprovider) hosted with ♥ by **GitHub (https://github.com)**

Na linha 24 temos o tratamento para os casos de CORS, pois imaginamos que teremos requisições para este end-point de vários domínios.

Feito este tratamento, das linhas 25 à 34 fazemos a validação do usuário e senha, e caso seja inválido, retornamos um erro (Linha 31).

Se tudo ocorrer bem, criamos um ClaimIdentity (Modelo de autorização) e retornamos como válido (Linha 40).

Com estes dois arquivos nossa API já está pronta para nos retornar um token.

Obtendo um token

Para obter um token, precisamos de três informações:

- grant_type
- username
- password

Neste caso, o grant_type será sempre password, pois estamos trabalhando com usuário/senha na autenticação, o que torna nossa requisição assim:

(https://aspblogs.blob.core.windows.net/media/andrebaltieri/WindowsLiveWriter/ImplementandoBearerAutenticationcomWebAP_C872/image_2.png)

Para a requisição acima utilizei o Postman, um App do Chrome.

Bloqueando Acessos

Para tornar uma action restrita a este modelo de autenticação, basta utilizar o atributo [Authorize] para decorá-la.

```
1  [Authorize]
2  [Route("")]
3  public IHttpActionResult Get()
4  {
```

```
    return Ok(new List<Customer>
    {
        new Customer(1, "André", "Baltieri"),
        new Customer(2, "Bruce", "Wayne"),
        new Customer(3, "Tony", "Stark"),
        new Customer(4, "Peter", "Parker")
    });
}
```

AuthControler view raw (https://gist.github.com/andrebaltieri/4035537809304c0df8e8/raw/2bceb3d63c00ebd9c2a5f4a40a2eb3892a7d1586/AuthControler) (https://gist.github.com/andrebaltieri/4035537809304c0df8e8#file-authcontroler) hosted with ♥ by **GitHub (https://github.com)**

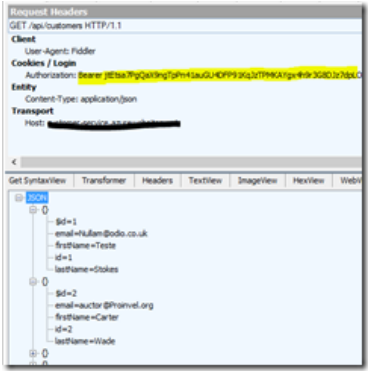
Simples né :)

Acessando um endpoint restrito

Agora, para acessar este conteúdo bloqueado, precisamos informar o tipo de autorização e token no cabeçalho da requisição.

(https://aspblogs.blob.core.windows.net/media/andrebaltieri/WindowsLiveWriter/ImplementandoBearerAutenticationcomWebAP_C872/image_4.png)

E como resultado temos:



(https://aspblogs.blob.core.windows.net/media/andrebaltieri/WindowsLiveWriter/ImplementandoBearerAutenticationcomWebAP_C872/image_6.png)

Um demo deste código está no meu GitHub: https://github.com/andrebaltieri/oauth-bearer (https://github.com/andrebaltieri/oauth-bearer)

Espero que tenham gostado!

André Baltieri
Microsoft MVP ASP.NET/IIS

http://andrebaltieri.net/ (http://andrebaltieri.net/)
http://facebook.com/andre.baltieri (http://facebook.com/andre.baltieri)