leastprivilege.com
Dominick Baier on Identity & Access Control

Embedding a simple Username/Password Authorization Server in Web API v2

Posted on November 13, 2013

In my last post I explained why I think it is important to use the authorization server pattern right from the start. In this post I want to show how to build the possibly simplest authorization server using the new Katana middleware that's shipping with Web API v2.

The scenario here is very similar to what I called "session tokens" before – the client sends a username/password to a token endpoint, and gets back an access token in return. Afterward the client can use this access token to communicate with the Web API.

The client can "forget" the user's credentials after the initial token request, and depending on the lifetime of the access token might not have to re-prompt the user for credentials for a reasonable amount of time. By adding refresh tokens in the mix you can also enable other interesting scenarios – but that's for another post.

We had an implementation for session tokens in Thinktecture.IdentityModel and we were able to layer that feature over arbitrary other authentication mechanisms (but most people used it really in conjunction with Basic Authentication). With the Katana OAuth2 middleware we will use the resource owner flow instead.

Writing an authorization server using Katana revolves around a class that derives from *OAuthAuthorizationServerProvider*. Here you can override various methods to take control over the OAuth2 protocol details. The minimum requirement though is to validate the incoming client in the *ValidateClientAuthentication* method..

Side note: when I say 'client' – I mean the client in the OAuth2 sense – which is the piece of software making the request [silicon-based lifeform] – not the human aka user of that client [carbon-based lifeform]. OAuth2 defines some mechanisms how the client can transmit credentials to the authorization server – this is really only suitable for certain scenarios – and not applicable to ours here. So we skip it (but will come back later to it).

```
public override async Task ValidateClientAuthentication
  (OAuthValidateClientAuthenticationContext context)
{
    // OAuth2 supports the notion of client authentication
    // this is not used here
    context.Validated();
}
```

Since we are technically speaking implementing the OAuth2 resource owner flow – the next method we need to implement is called *GrantResourceOwnerCredentials*. In this method you need to validate the username and password. If that succeeds, you create a *ClaimsIden* epresents the authenticated user. Any claims you add to this identity will become pa ıus be accessible by the Web API.

Typically the claims you add descı e.g. his user id (the 'sub' claim) or roles he is member of. Don’t go crazy here! T bigger the token gets – and the token has to be submitted on every request.

```
public override async Task GrantRes
  (OAuthGrantResourceOwnerCredentia
{
    // validate user credentials (c
    // user credentials should be stored securely (salted, iterated, hashed…)
    if (context.UserName != context.Password)
    {
        context.Rejected();
        return;
    }

    // create identity
    var id = new ClaimsIdentity(“Embedded”);
    id.AddClaim(new Claim(“sub”, context.UserName));
    id.AddClaim(new Claim(“role”, “user”));

    context.Validated(id);
}
```

…and that’s really it for a bare-bones resource owner credentials flow. The next step is to wire up the authorization server to the Katana pipeline. There are two separate pieces of middleware that you need. The authorization server middleware takes care of handling the token request and generation – the bearer token authentication middleware for consuming the token:

```
public void Configuration(IAppBuilder app)
{
    // token generation
    app.UseOAuthAuthorizationServer(new OAuthAuthorizationServerOptions
        {
            // for demo purposes
            AllowInsecureHttp = true,

            TokenEndpointPath = new PathString(“/token”),
            AccessTokenExpireTimeSpan = TimeSpan.FromHours(8),

            Provider = new SimpleAuthorizationServerProvider()
        });

    // token consumption
    app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());

    app.UseWebApi(WebApiConfig.Register());
}
```

The *AccessTokenExpireTimespan* property specifies how long the access token should be valid. Keep in mind that there is no built-in way to revoke access tokens once they have been issued. So if you want to disable the user or update claims, that’s the latency you will have. Again refresh tokens can help here (future post).

The Client

In the client you need to divide your logic into two parts – request the token (sending credentials) and use the token.

Since requesting the token is using normal OAuth2 resource owner flow, you can use some OAuth2 client or you handcraft the HTTP request yourself (see the spec). Using the [Thinktecture.IdentityModel.Client](#) *OAuth2Client*, this boils down to:

```
private static TokenResponse GetToken()
{
    var client = new OAuth2Client(
        new Uri("https://localhost:2727/token "));

    return client.RequestResourceOwnerPasswordAsync("bob", "bob").Result;
}
```

…and using the token could look like this:

```
private static void CallService(string token)
{
    var client = new HttpClient();
    client.SetBearerToken(token);
    var response = client.GetStringAsync(
      new Uri("http://localhost:2727/api/identity ")).Result;
}
```

The full sample can be found [here](#).

Share this:

Google Twitter 20 Facebook 3 Pinterest Tumblr Pocket

★ Like

2 bloggers like this.

Related

[Adding Refresh Tokens to a Web API v2 Authorization Server](#) In "AuthorizationServer"	[Dissecting the Web API Individual Accounts Template–Part 2: Local Accounts](#) In "ASP.NET"	[Authorization Servers are good for you (and your Web APIs)](#) In "AuthorizationServer"

This entry was posted in Uncategorized. Bookmark the permalink.

38 Responses to *Embedding a simple Username/Password Authorization Server in Web API v2*

Pingback: *[Adding Refresh Tokens to a Web API v2 Authorization Server | www.leastprivilege.com](#)*

[Nick Meldrum](#) *says:*

November 22, 2013 at 17:47

Excellent article Dominic thanks – just one note, unless your client software is a breast enlargement, you probably meant silicon-based lifeform, instead of silicone ;) /pedant

Reply

Dominick Baier *says:*

November 26, 2013 at 08:51

;)

Reply

Pingback: *Dissecting the Web API Individual Accounts Template–Part 2: Local Accounts | www.leastprivilege.com*

Tugberk Ugurlu *says:*

December 23, 2013 at 16:28

Very great post! Thanks a lot. One lazy question:

How is the access_token protected by default by the katana middleware? We can configure it as far as I know but what is the default behavior?

Reply

Dominick Baier *says:*

December 23, 2013 at 16:31

Depends on the host. in ASP.NET they use the machine key. otherwise they fall back to DPAPI. Or you provide a mechanism.

Reply

bartek4c *says:*

January 24, 2014 at 19:16

Great post. One question though, since its OAuth2 implementation that you present and bearer token that is being used, there is no other way to secure token in transfer apart of using SSL, am I right? Therefore this scenario is at risk of MITM attack and should not be used for critical data access?

Reply

Dominick Baier *says:*

January 26, 2014 at 11:09

You are right – typically OAuth2 uses bearer tokens – you could implement that differently of course – I mean as much bearer as the original password ;)

But not sure I agree with you conclusion – if SSL cannot be trusted to provide security – then bearer tokens become your smallest concern.

Reply

bartek4c *says:*

January 27, 2014 at 10:45

Thanks for your answer. I just work for the company that cannot afford SSL and am considering possible options for security

shahramsoft1 *says:*
January 27, 2014 at 09:10

Very great Article .I Have Lazy Question :
What should i write in my Api To check The Token? because I have Some Api's which i want to call from my own Program as well as Out Side Of my Program.

Reply

Dominick Baier *says:*
January 27, 2014 at 10:39

What do you mean with "check" – the token is automatically validated by the middleware. You get access to the claims via RequestContext.Principal.

Reply

Dominick Baier *says:*
January 27, 2014 at 10:54

Well without SSL you have to built your own server authentication, integrity protection, confidentiality protection and replay protection…that's not worth it.

I don't buy the "can't afford" argument.

Reply

Poul K. Sørensen (@pksorensen) *says:*
February 20, 2014 at 02:11

You can get a free SSL certificate at startssl ect, and for 60$s you get your identity validated for wildcard domains. So "can't afford" is a little weak. By the time you read this message it would have been cheaper to buy the SSL certificate then to find an alternative :)

Reply

Pingback: *The Web API v2 OAuth2 Authorization Server Middleware–Is it worth it? | leastprivilege.com*

Pieter Dhondt *says:*
March 25, 2014 at 18:42

We have two different Web API's and an MVC Web App that we want to secure.
As clients we have a MVC Web App, javascript code in this, and a separate desktop client.
And we have a user database which can validate users.

Is it possible to access the MVC Web app and the Web API's from Javascript with the same access token? And similarly from the desktop client.

Can we set up a central authorization server that grants these access tokens (checking against our user db) which then all 'connected' clients (both Web API's and the MVC Web App) can consume?

Do you have a sample/article to learn from which covers more or less this scenario? :)

Reply

Dominick Baier *says:*

April 1, 2014 at 08:06

Well – Web apps typically use cookies, APIs use tokens. They don't go together so well, and need some thought.

AuthorizationServer would be such a central component:
https://github.com/thinktecture/Thinktecture.AuthorizationServer

Reply

The Anachronist (@rnarayana) *says:*

March 31, 2014 at 19:23

Thank you! Could you tell where JWT comes into the picture here? Especially the JwtSecurityToken class and all. Is it a wrapper over OAuth? Why?

Reply

Dominick Baier *says:*

April 1, 2014 at 08:04

No JWT – Microsoft's embedded AS uses a proprietary token format. It was meant to simplify OAuth2.

Reply

mvbaffaMarcus Baffa *says:*

April 9, 2014 at 04:59

Hi Dominick,

I am working with Web APi 2 and Owin and bearer access token authentication. I have a problem that I could not determine the solution.

Is there a way to reset a password ?

Thanks in advance

Reply

Scott Morman *says:*

April 23, 2014 at 02:31

Great article! Thank you! Is there a way to override the AccessTokenExpireTimeSpan? I have tried setting the AuthenticationProperties.ExtireUtc value while the access token is being created, but the expires_in value always seems to come back to the client with the AccessTokenExpireTimeSpan value that was configured. I would love to be able to override this value and set the expiration based on the client_id making the request. Any ideas on how to do that?

Reply

Dominick Baier *says:*

April 24, 2014 at 06:48

I haven't tried it yet – but wouldn't be surprised if the MS middleware is not flexible enough for that. If I find the time I will do some research.

See also here: http://leastprivilege.com/2014/03/24/the-web-api-v2-oauth2-authorization-server-middlewareis-it-worth-it/

Reply

---

ty black *says:*
May 22, 2014 at 04:11

Dominick Thank you so much for bringing light to this subject as i have just started learning "Web programming" 2 weeks ago I have a tendency to ask, dumb questions.

From what i gather from your second paragraph "the client sends a username/password to a token endpoint, "
Am i to assume that this end point is running on the same server as all the other web api endpoints?
I ask this b/c i watched your video on this topic and i thought that these 2 things, (1. the token giver, and 2.all the other web api stuff,) were suppose to be in separate processing or even separate servers and not just separate api routes on the same process.
thanks
Ty

Reply

Dominick Baier *says:*
May 23, 2014 at 08:33

That's up to you. You can start small and "embed" the token endpoint into the API itself – or use a separate service / server for it. The important part is, that the token endpoint is logically separated from your business logic.

Reply

Taiseer Joudeh *says:*
July 5, 2014 at 05:19

Hi Dominick,
Quick question, is there any way to capture the generated token before it is sent in the response to the client? I want to store the token in localdb and I'm looking for an event after we call context.Validated(ClaimsIdentity);
Thanks,
Taiseer

Reply

Brad *says:*
August 26, 2014 at 16:32

Hi Dominick,

I have a situation where I have a Web Api 2 controller that is in APP (A). A client is installed via a NuGet pkg that other trusted realms use to access this functionality. What I am trying to accomplish is this:

I know that only our own trusted apps can install and use this client. Users in our system are granted access to 1 or more apps that are secured via our STS. Where this client is concerned, a user may or may not have been granted access to this APP (A).

However, for this certain functionality in APP (A), I don't want to care if a user in APP (B) or APP (C) etc. has been granted specific access to APP (A). I only want to care if a user is authentic and has been granted access to "at least 1" app in our system.

I don't want to create a new "role" in APP (A) and I don't want to give all users access to APP (A) by default. First, is this even a good idea? And second, I'm not sure what the best way is to implement this. Can you give me any ideas or point me to an article that covers this?

Thanks in advance.

Reply

Dominick Baier *says:*

August 27, 2014 at 18:49

I am not sure I follow you ;)

Reply

JimmyB *says:*

September 9, 2014 at 18:21

Hi thanks for the tutorial ,

Is there a way to reset the password by a reset link or any other way?

Reply

Dominick Baier *says:*

September 9, 2014 at 18:52

That would be up to user management – I am only describing the token issuance part.

Reply

paolo *says:*

October 16, 2014 at 10:55

Great tutorial , but i need to have one web api for authentication(A) e one for data(B), i have tried but when call the B controller from an angularjs app the user is authenticated for the web app but not for the api B: how to inform B that user is authenticated by A?
Thanks

Reply

Dominick Baier *says:*

October 16, 2014 at 13:49

By using an authorization server e.g.

https://github.com/thinktecture/Thinktecture.IdentityServer.v3

Reply

eugenekgn *says:*
November 21, 2014 at 05:07

Hey Dominick,

Is it possible to get credentials back with the token without making two calls first to create toke via /token and then /api/identity because it seems a bit redone-dent to make two calls in my case given that I have all the information needed during authentication and all I want to do is pass it back.

Thank you
Eugene

Reply

Dominick Baier *says:*
November 21, 2014 at 20:53

Well – the access token is not meant for the client.

Reply

Paul *says:*
January 25, 2015 at 09:43

Hi Dominick, your explanations of this are ivery helpful, I'm not sure I would have known what took do otherwise!

You mention machine key in an earlier comment. Can you confirm that the default, out of the box, behaviour of OAuthAuthorizationServerProvider is to protect the refresh token with a machine key so that a token generated on another machine can't be used to gain access? Is that machine key automatically generated or do I need to define it?

Reply

Dominick Baier *says:*
January 25, 2015 at 12:27

Access tokens are protected using the machine key. Protection of refresh tokens is up to you. You need to manually set the machineKey element in web.config (otherwise the key gets auto generated and might change over time).

Reply

Justin Maier *says:*
February 25, 2015 at 02:58

Hey Dominick, thanks for this great example, it was very helpful.

I'm having trouble with user.IsInRole() always returning null. Is there another step to making that work?

Reply

Dominick Baier *says:*
February 25, 2015 at 22:27

You probably need to set the role claim type on the token consuming middleware (or use the lame ClaimTypes.Role instead when producing the token).

Reply

Justin Maier *says:*

March 2, 2015 at 20:26

Thank’s Dominick you were exactly right. I also has to change ‘sub’ from your example to ClaimTypes.Name thanks for taking the time to reply.

N *says:*

August 6, 2015 at 20:19

Dom, not sure how to ask this question. I’m researching ways on how to have a web api sit on a corporate network. People will use mobile devices to make web api calls. These people should be part of the domain and need to be authenticated before making web api calls. I think this screams claims identity, session tokens, etc. Can you recommend any good resources?

Reply

leastprivilege.com

The Twenty Ten Theme. *Blog at WordPress.com.*