

## Re: Custom authentication in a web application

**Source:**

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.webservices/2004-10/0174.html>

---

**From:** Tom Porterfield (*tporter\_at\_mvps.org*)

**Date:** 10/13/04

Date: Wed, 13 Oct 2004 14:29:01 -0400

Jon Skeet [C# MVP] wrote:

> *(Thanks ever so much for the reply, btw. It's a good start for me :)*  
>  
>> *We do this using Windows Authentication with our security data stored in  
>>a SQL server database. On the server we create our own principal object  
>>that inherits from WindowsPrincipal.*  
>  
> *Any reason for using Windows Authentication here rather than any of the  
> other types?*

This is an internal app where we have complete control of the workstation configuration where we would be called from. We don't necessarily know who the client is, or will be in the future.

WindowsAuthentication for web applications is company policy.

>>*In Global.asax in the  
>>AuthenticateRequest handler we replace the HttpContext.Current.User with  
>>our principal object, passing HttpContext.Current.User.Identity as  
>>WindowsIdentity to the constructor. Our principal object overrides the  
>>two overloads of IsInRole to use our own security check. We have also  
>>added a HasPermission method to our principal so we can demand a  
>>permission whenever we need to. So our AuthenticateRequest handler  
>>looks as follows:*  
>>  
>>*protected void Application\_AuthenticateRequest(Object sender, EventArgs e)*  
>>*{*  
>>*// this will throw an exception if windows auth not turned on*  
>  
>  
> *How does the behaviour differ between the situation where the user  
> actually \*is\* a valid Windows user for the system, and where they're  
> not? Isn't ASP.NET or IIS going to have tried to use whatever the  
> client provides as Windows authentication by now?*

Yes, so effectively there are two levels of security involved here. But since we may not know who our client is, we don't want to rely on the fact that they have properly authorized the user. Right now you simply

have to be authenticated to a trusted domain to execute our web services. Another option would be to create a domain group and only folks in that domain group have access. By using windows authentication that is possible, but again the scenario here is we want to use our own security database. If in the future we wanted to use true windows authentication, or active directory, our custom principal object would need to change but nothing else in the application would need to change.

When you know your clients are all at least running Windows, this seemed like the logical future-protection mechanism.

```
>> // Also note we have to set the context here since ASP.NET will take
>> // what's in the context and place it on the Thread.CurrentUser property
>> CustomServerPrincipal princ = new CustomServerPrincipal as
>> WindowsIdentity);
>> HttpContext.Current.User = princ;
>
>
> Any reason for doing it as WindowsIdentity rather than just setting it
> as a CustomServerPrincipal?
```

The code didn't get in there properly. The constructor of WindowsPrincipal takes a WindowsIdentity as the parameter. The line should be:

```
CustomServerPrincipal princ = new
CustomServerPrincipal(HttpContext.Current.User.Identity as WindowsIdentity);
```

```
> Presumably before setting the value you check whether the user/password
> combination is valid?
```

Since all of our users are internal, the fact that they have successfully logged on to our domain means they are validated. And again, maybe that is a difference here that would require a different approach for you. So no need to check that they are valid. Rather we only need to check to see if their ID has authority to our service.

```
>> // verify that this user is authorized to get into Polaris
>> if (!princ.HasPermission(authUserPerm))
>> {
>> throw new
>> CustomSecurityException(String.Format(securityExceptionMessage,
>> princ.Identity.Name));
>> }
>>}
>
>
> Does the type of exception matter here, out of interest?
```

We have encapsulated the exceptions so as to capture and present the exception information in a way that is consistent for our services. So from a generic standpoint, no it doesn't matter.

microsoft.public.dotnet.framework.webservices: Re: Custom authentication in a web application

>>In any server side objects where we need to demand a permission, we now  
>>simply take the current principal from the thread as our custom  
>>principal and demand the permission. Ex:

>>

>>CustomServerPrincipal principal =

>>System.Threading.Thread.CurrentPrincipal as CustomServerPrincipal;

>>if (!principal.HasPermission(deletePermission))

>>{

>> throw new CustomSecurityException(principal, deletePermission);

>>}

>

>

> Right – that bit I think I'm reasonably happy with.

>

--

Tom Porterfield