# Web Request Authentication

## WCF REST Services hosted in IIS

WCF services can be hosted in IIS and activated on demand. This makes a separate hosting process unnecessary and the services might benefit from ASP.NET integration, namely have access to the HttpContext including the application, session, request, response objects etc. This can be achieved by turning ASP.NET integration in by specifying the following in web.config:

<system.serviceModel>   
<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" />   
</system.serviceModel>

Using ASP.Net integration has it’s overhead and prevents reusing the WCF service outside IIS, so Graywulf implements REST Services without integration but are hosted within IIS.

To host a WCF service in IIS, a Service1.svc file needs to be created with the following content:

<%@ ServiceHost Language="C#" Debug="true" Service="Jhu.Graywulf.Web.Api.V1.DataService" Factory="System.ServiceModel.Activation.WebServiceHostFactory" %>

Here the name of the factory class is very important, this determines the type of bindings that are created when the service class is instantiated. WebServiceHostFactory creates webHttpBinding objects that support REST communication (automatic response serialization format selection, etc.) instead of SOAP.

WCF services can also be hosted in IIS by explicitly registering them somewhere in the code (most likely in AppStart) the following way. First the URL routing module needs to be added to the configuration:

<httpModules>   
<add name="WebAuthenticationModule" type="Jhu.Graywulf.Web.Security.WebAuthenticationModule, Jhu.Graywulf.Web"/>   
<add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Routing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />  
</httpModules>  
<httpHandlers>  
<add verb="\*" path="UrlRouting.axd" type="System.Web.HttpForbiddenHandler, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />  
</httpHandlers>

Then the services can be started the following way. The first line is required to exclude compiled resource from under URL routing.

RouteTable.Routes.Ignore("{resource}.axd/{\*pathInfo}");  
RouteTable.Routes.Add(new ServiceRoute("Api/Jobs/", new WebServiceHostFactory(), typeof(Jhu.Graywulf.Web.Api.JobsService)));

## Web Forms authentication

Web forms authentication works by sending a ticket in the form of cookie to the client. The ticket is signed and encrypted by the web service, so it can be read only by the server itself, or by any other server that is in the possession of the machine key. Tickets contain information on the identity of the user, an expiration time and a few other settings.

The advantage of using these tickets is that users can be identified simply by decrypting the ticket and there is no need reading any database or making requests to identity providers.

Because this authentication method depends on cookies, the client has to support it (web browsers do it by default, whereas REST clients need to be written such a way). Cookies aren’t valid across different domains, so all websites sharing the ticket should be under the same domain, otherwise this method cannot be used for single sign-on, unless the user is redirected to the sign-on page every time a web site under a different domain is accessed.

Web Forms authentication tickets can be used even when users are not authenticated via a web form but via tokens, or other means of authentication (for example OpenID or Kerberos). The advantage of this is that the users’ identity can be kept encrypted and saved across web applications and sessions.

## Token-based authentication (Keystone)

## Response-based authentication (OpenID)

## The authentication model

In Graywulf, authentication modules are provided for web sites accessed by users and REST web services accessed by software clients. The two scenarios differ in a few ways:

WebAuthenticationModule can use the HttpContext to get access to the headers of the HTTP request and has access to the HttpApplication and Session objects. It implements the IHttpModule interface to integrate with the ASP.Net framework. Turning on the module requires the following line to exist in web.config:

<system.webServer>  
<modules runAllManagedModulesForAllRequests="true">  
<add name="WebAuthenticationModule" type="Jhu.Graywulf.Web.Security.WebAuthenticationModule, Jhu.Graywulf.Web"/>  
</modules>  
<add name="UrlRoutingHandler" preCondition="integratedMode" verb="\*" path="UrlRouting.axd" type="System.Web.HttpForbiddenHandler, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />  
</handlers>  
</system.webServer>

RestAuthenticationModule, on the other hand, can only use the WebOperationContext to gather information about the request. Instead of deriving from AuthenticationManager, the REST module implements the IDispatchMessageInspector and IParameterInspector interfaces that provide function which are called before and after service functions are invoked. This differs from the typical WCF way of authenticating requests, but is more similar how HTTP works, consequently many functions can be shared between modules designed specifically for the web user interfaces and REST services.

Due to the two different ways of authentication, Graywulf provides the AuthenticationRequest class to make access to the request variables transparent.

## Web site request authentication life-cycle

1. A request to the server is made by the client
2. If this is the first request no session cookie, or forms authentication ticket exists. A token in the request might exist, however, which identifies the user.

## Scenarios to be tested

1. New user arrives, no cookies, no tokens
2. User arrives, no cookies, but token exists
3. New user arrives, but old session exists