EMC²
where information lives®

EMC Documentum Single Sign-on Using Standard Tools

Sander Hendriks

EMC Proven Professional Knowledge Sharing 2010

EMC² PROVEN PROFESSIONAL

Sander Hendriks
Documentum Architect
Informed Consulting
sander.hendriks@informedconsulting.nl

Table of Contents

Disclaimer: The views, processes or methodologies published in this compilation are those of the authors. They do not necessarily reflect EMC Corporation's views, processes, or methodologies

Introduction

Single Sign-on (SSO) is every system users dream. Imagine starting your PC in the morning, logging in once, and never worrying about your password for the rest of the day. All the applications will be informed of your identity securely so they can authorize you for the data and functionality you need. Everyone wins: the users, the applications, and the network security team.

Then why are Documentum® applications in most organizations still displaying a login screen when you start your browser? The technical implementation of Single Sign-on for a web application can be difficult to implement, as I've learned from personal experience. I've struggled through several ways to implement SSO and the good news is… it works!

The road to successful Single Sign-on can be long and bumpy. There are technology choices to make, products to configure, integrations to be tested, maybe even code to be written. The purpose of this article is to guide you through this process so you may avoid some of the pitfalls and benefit from the lessons learned from my experience.

I will describe the different SSO implementations that you can use with Documentum and mention some of their advantages and disadvantages so you can decide which is best.

Then I will dive into the details of a Kerberos implementation. I'll show several ways that a Documentum application can make use of Kerberos.

I hope this will help anyone facing an SSO implementation to make the right choices. Maybe you'll one day be asked to reply to a pleasantly surprised user who asks "How did the application know it was me logging on?"

The Basics of Single Sign-on

Your goal is to use Documentum applications without requiring users to type in their username and password every time they open their browser.

But how does that work?

Documentum applications are usually implemented as web applications, so there are four main parts of architecture:

- The user's browser
- The application server
- The Documentum content server
- A server holding the list of users and their credentials

Getting the user's identity all the way from their PC to the content server securely, leaving no opportunity for impersonation or other abuse, is the difficult part.

So what happens on the client side?

1. The user comes in to the office in the morning and starts his PC
2. He enters his username and password
3. The PC contacts the directory server to check the user's name and password
4. If all is well, Windows is started and the user can begin working
5. He starts a browser and enters an address similar to http://appserver.domain/dctmapp

At this point, we need some 'magic' to get authenticated on the app server and to make the app server start a session with the repository on the content server for the right user. A central Identify Server will help us to accomplish this. The idea is simple; the user identifies himself to the Identity Server once. The Identity Server acts as a trusted third party. Whenever the user wants to access a service, he will pass his identity to the service and the service can check that the identity is valid with the Identity Server.

All applications that need to support Single Sign-on are registered on the Identity Server, as well as all users who need to access those applications. The Identity Server communicates with client PCs and servers using encrypted messages to prevent forgery. It stores public and private keys for all registered servers and users.

What happens is shown in the following illustration:

App server authentication

Content server Authentication

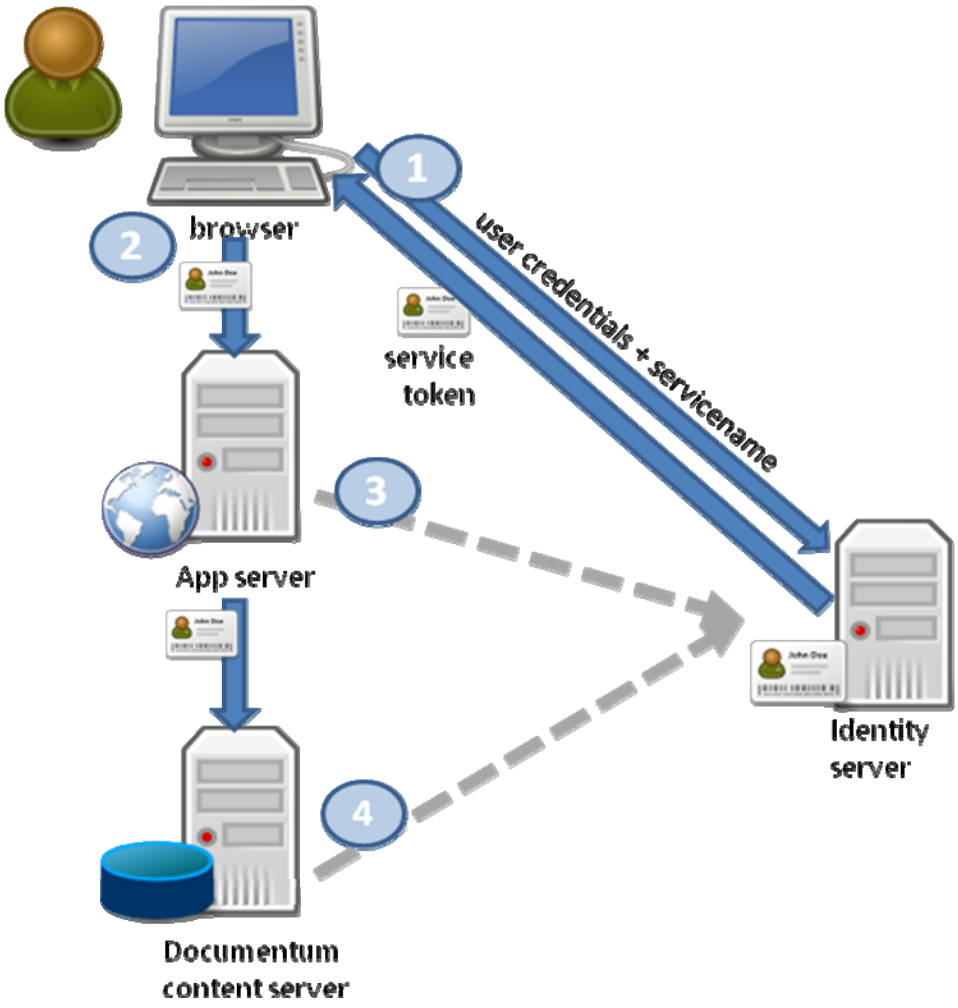

Fig. 1: Generalized SSO authentication process

1. The user identifies himself to the Identity Server through a username and password, or a hardware token, fingerprint, or some other means of authentication

2. He then surfs to http://appserver.domain/dctmapp. A service token is retrieved from the Identity Server for the dctmapp service and sent with the request to the App Server

3. The App Server checks the service token. How this is done depends on the SSO solution used; it may involve decrypting the service token, or contacting the Identity Server to verify the user's identify.

4. The App Server then passes the service token to the Documentum Content Server, that

5. can then verify the user's identity and start a Content Server repository session

SSO Options for Documentum applications

There are several ways to achieve Single Sign-on for Documentum applications:

- Using a specialized product, such as RSA or CA SiteMinder
- Using a third-party Kerberos implementation, offered by Solfit
- Using Kerberos and a trust relation between the application and the Content Server
- Using Kerberos all the way to the Content Server

All of these options have advantages and disadvantages that I will describe below.

Using a specialized product

Documentum is integrated with two well known SSO products:

- RSA Access Manager http://www.rsa.com/node.aspx?id=1186
- CA Siteminder http://www.ca.com/us/internet-access-control.aspx

These two product integrations are supported by EMC and are well documented in the Content Server Admin Guide. The integration with both products is similar, though the underlying authentication technology is different.

You need the following to make it work:

1. Install the RSA Access Manager or CA Siteminder on your network

2. Install an authentication plug-in for your selected product on the Documentum Content Server. This will enable it to authenticate users when they supply an SSO security token instead of a password

That's it. This is provided as an out-of-the-box integration by Documentum, so if you have your SSO product set up correctly, it will work.

There are a few other products in this category:

- IBM supports SSO integration of Documentum with Tivoli Identity Manager
- SAP supports SSO integration of Documentum with SAP

Advantages

+ Rely on an SSO product that is built for the job
+ Works with only minor configuration needed in Documentum

Disadvantages

- The SSO product has a purchase and maintenance cost
- The SSO product adds an extra server to your network
- The SSO product must be installed, configured, and administered

Using a third-party Kerberos implementation

Solfit, a Swiss-based IT company, provides a SSO solution for Documentum applications. The solution is based on Kerberos (that will be discussed extensively below) and uses the Active Directory that the users are already logging on to as the Identity Server.

You only need to do the following:

1. Order Documentum SSO from Solfit http://www.solfit.ch/solfit_documentum_sso.php
2. Solfit will send a consultant to install and configure their product

You're done. The consultant may need to return to install the SSO solution on your production system later, though.

Advantages

+ Rely on a solution especially created to enable SSO for Documentum
+ The solution supplier takes care of all configuration
+ It makes use of products you already own: Documentum and Active Directory

Disadvantages

- The SSO product has a purchase and maintenance cost
- You will be charged a consultancy fee for the installation and configuration

Using a custom Kerberos implementation

Consider creating your own SSO implementation. You don't have to create a SSO product from scratch; you can use your Active Directory server and the SSO integration options provided by Documentum. You have a choice on how far you want to take things. There is an "easy" way (less secure) and a more involved way (more secure). More about that in the next section.

For your own Kerberos SSO integration you will need:

1. An Active Directory server, or another Kerberos server
2. IT consultants who know Documentum and the Kerberos protocol
3. Some time and effort for a Documentum-Kerberos integration project

Advantages

+ No out-of-pocket cost for SSO products
+ It makes use of products you already own: Documentum and Active Directory

Disadvantages

- You will need consultants who know Documentum and Kerberos
- Implementation project and maintenance on custom code

Selecting the best SSO option for your project

The options for SSO integration with Kerberos are very different: a specialized product, a consulting solution, or a customization project. Which is best depends on your circumstances. Here are some pointers:

- A specialized product may be best for you if you already operate a SSO product from RSA, CA, IBM, or SAP, or if you plan on adding Single Sign-on to your wider system landscape, to be used by Documentum as well as non-Documentum applications

- The Solfit solution may be best if you want a Documentum SSO integration that comes as a complete package including installation and configuration services, with no need for your organization to invest in knowledge of the technology used

- The custom solution may be best if you don't want extra servers or protocols in your network, if you want to minimize your reliance on outside suppliers, or if you don't have the budget to invest in SSO products.

Kerberos SSO

Since the Single Sign-on offerings by RSA, CA, and Solfit are extensively documented, the rest of my article will focus on the custom Kerberos solution. It will show that there are several viable options for implementation and will offer some pointers and best practices.

Kerberos history

Kerberos is an authentication protocol that was designed by John Kohl and Clifford Neuman from MIT in 1993 (see http://en.wikipedia.org/wiki/Kerberos_%28protocol%29). They designed it for use in Project Athena, a distributed educational network.

It is an open standard, supported by the Internet Engineering Task Force (IETF) as described in RFC 4120: http://tools.ietf.org/html/rfc4120

The protocol was picked up by large IT vendors. Microsoft added support for Kerberos to Windows XP and Windows 2000 Active Directory as a replacement for the NTLM protocol that had been used since Windows NT. Currently, there is a Kerberos implementation for almost any platform, including Windows, UNIX, and Linux.

More information on Kerberos:
http://technet.microsoft.com/en-us/library/cc772815%28WS.10%29.aspx

Internet standards

There are a few other IETF standards that are relevant to a Kerberos implementation. **GSSAPI** (Generic Security Services Application Programming Interface) is a standard that describes an interface to security libraries, so that applications can use security services in a uniform way, independent of the underlying authentication protocol. GSSAPI libraries exist for many operating systems and programming languages.

SPNEGO (Simple and Protected GSSAPI Negotiation protocol) is a standard mainly used by Microsoft to enable Windows to support multiple authentication protocols, so that old Windows NT clients can connect to new Active directories, and new Windows7 clients can connect to old Windows LAN Manager domains.

SPNEGO is a negotiation protocol that clients and servers use to communicate about the authentication protocols they support. This helps them select the best authentication method supported on both sides. SPNEGO supports Kerberos and NTLM as authentication methods, where Kerberos is the preferred method and NTLM is the fall-back method if Kerberos is not supported by either the client or server.

SPNEGO was implemented in Internet Explorer 5 and IIS 5 to add SSO authentication ability to web sites and web applications. In Internet Explorer it is known as ‘Integrated Windows Authentication’. SPNEGO is also supported by Mozilla Firefox and some other browsers.

How does Kerberos authentication work for Documentum applications?

Kerberos authentication of web applications is performed in two phases:

1. There is interaction between the user's browser and the application server to authenticate the user
2. There is interaction between the app server and the content server to start a DFC session with the Documentum repository

Both phases will be discussed in detail in separate paragraphs below.

One of the strengths of the Kerberos protocol is that NO interaction with the KDC is needed to authenticate a ticket. The user is authenticated if a ticket can be decrypted and the authenticator inside the ticket checks out.

The authentication process is shown in the illustration below:

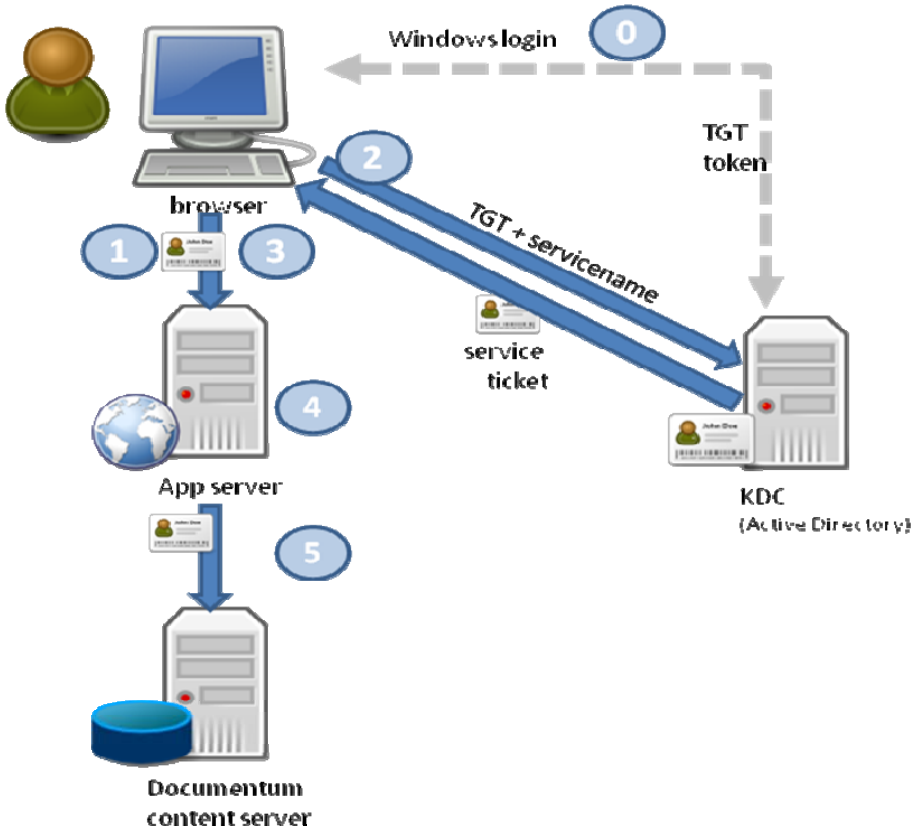

Fig. 2: Kerberos SSO authentication process

0. The user logs in to the Kerberos Key Distribution Center (KDC); for Windows environments the Active Directory acts as KDC, so logging in to the Active Directory Domain will log you in to its KDC. On successful login, the KDC will respond by sending the client a TicketGrantingTicket (or TGT). This will be used later for retrieving service tickets for services.

1. The user surfs to http://appserver.domain/dctmapp; the browser will at first send a plain HTTP GET, or POST; the application server realizes that user authentication is required for this request and will respond with an authentication error; the response will contain a header 'Negotiate'.

2. The browser will realize that SPNEGO authentication is required; it will send a request for a service ticket for service HTTP/appserver@AD.DOMAIN to the KDC.

3. The browser will resend the request to the App server, adding the service ticket to the request headers.

4. The App server checks the service ticket and processes the user's request.

5. The App server then passes the service ticket to the Documentum content server that verifies the user's identity and starts a content server repository session.

Tip: For SPNEGO to work, your web app server must be registered on the KDC/Active Directory with the correct Service Principle Name (SPN)

If the user is logged in to domain ABC.COM and calls http://server123.abc.com, then the browser will request a service ticket for SPN HTTP/server123@ABC.COM.

There is no way to change the browser behavior, so you must register your app server with this SPN, or SPNEGO won't work.

Adding SPNEGO support to your web application

What do you need to enable SSO authentication between the browser on the client PC and the web application server? You need SPNEGO.

Get a browser that supports SPNEGO

Internet Explorer and Firefox, the most popular browsers, support SPNEGO.
For SPNEGO to work in Internet Explorer, there are two conditions:

- You need to enable 'Integrated Windows Authentication' in the Internet Options
- The application should be in the 'Intranet Zone' of IE, so you may need to add the URL of the Documentum application to the Local Intranet Zone sites on the Security tab of IE's Internet Options

Get a web application server that supports SPNEGO

This may be a bit more involved than the client side. While all big commercial application servers – like WebSphere, WebLogic, and JBoss – support SPNEGO, they all support it in their own way, so you'll need to consult the product documentation to find out what you need to do to set up your App server for SPNEGO authentication.

> Tip: Make sure that the application gets access to the username (from the SPNEGO service ticket); you'll need it in the next chapter.

Another popular and free App server, Tomcat, does not support SPNEGO out-of-the-box. There are several ways around this:

- You can use Tomcat's security system and add a 'Valve' that takes care of SPNEGO for you. There is a SpnegoValve available on the net. I had some trouble getting it to work, but it may be just the thing for you.

- You could write your own Java SPNEGO code using Java's built-in security libraries: JAAS and JGSSAPI. There are several good samples available as a starting point on the net. Be aware that you'll need Java version 6 if you go this road since most existing Documentum systems use Java 5. This may entail a migration from Java 5 to Java 6

(Documentum products support Java 6 since 6.5, so this is mostly an option for new projects that use the newest versions of all products involved).

- Add an Apache proxy. This is an interesting option if you want to add SPNEGO to Tomcat with no Java coding. The Apache web server supports SPNEGO through its mod_auth_kerb module.

The last option merits a bit more explanation. Adding Apache will mean that the browser will no longer communicate directly with the application server. It will talk to the Apache web server, Apache will take care of the SPNEGO authentication, and will then pass the request to Tomcat. The response will go back to Apache that will pass it back to the client browser.

Advantages

+ There is no Java coding involved, just configuration
+ Apache can be run on port 80 (the default HTTP port), so the users don't need to specify a port number in the application URL
+ Administrators can still use Tomcat on the original port 8080 to get to the Documentum application without using SSO (they will get a login screen)
+ Apache can be configured to do some other useful things

Disadvantages

- An extra component is introduced in your network architecture

Tip: Configure Apache to add the username to a request header; this can be done with mod_rewrite and mod_header.

Tip: Not all URLs should be protected with security; the applets used by Documentum applications don't support SPNEGO, so they should be exempt from authentication.

Communication between the web application server and the content server

The web application server has received a request and it's been authenticated. Now it is time to get the data, produce some HTML, and send a response back to the browser.

The data needs to be retrieved from a Documentum repository, so the application server needs to set up a DFC connection to the content server that serves that repository. Since security and accountability are critical in any serious Documentum implementation, we can't just have one server connect to the other using a service account as the app server would connect as 'admin' to the repository for all user requests. All queries would be performed by user 'admin', the audit trail would be full of 'admin', and all objects would be owned by 'admin'. This is not a good idea.

Open a DFC session for the user that placed the web application request. There are two viable ways to do this:

- Use Documentum ticketed login
- Use Kerberos all the way to the content server

Ticketed login

Trusted login is a Documentum feature especially designed for cases where one back-end machine needs access to another. It works as follows:

> If you have a connection to a Documentum repository and have super user privileges, you can get a login ticket for any other user. That ticket can then be used to make a connection on behalf of that user.

The catch is that you need a super user session before you can make sessions for regular users. Super user sessions can be made in several ways, but almost all of them involve storing the super user's credentials somewhere on the app server. That is the weakness of this mechanism. If an intruder can gain access to the credentials, he can connect to the Documentum repository as super user and compromise the system. You can minimize this vulnerability with adequate security measures, such as protecting the credentials with encryption and minimizing access to those credentials to only the account that is running the app server.

When you consider this way of connecting, estimate the risk of abuse in your environment. If you don't want to run that risk, use Kerberos to the Content Server instead.

Implementation of Ticketed Login requires some customization of the login function of the web application. Some code should be written to do the following:

- Log in to the repository using super user credentials
- Get the user's name from the request (usually from an HTTP header)
- Retrieve a login ticket for that user
- Create a repository session for the user, with the login ticket as password

Some customers have implemented Ticketed Login for IBM Websphere application server.

Kerberos on the Content Server

Kerberos is a standard and secure protocol for user authentication, so why not use it all the way to the Content Server? The answer used to be because Documentum does not support Kerberos authentication out-of-the-box. However, things are changing on this front:

Tip: Documentum will be supporting Kerberos authentication on the Content Server in the upcoming 6.6 release.

For those of us who are unwilling to wait that long, the good news is that you don't have to. The Content Server has an Authentication Plug-in API that allows you to write your own custom authentication mechanism. This API can be used to write a Kerberos authentication plug-in. The downside is that the API is written in C++, so your average Documentum Java programmer is up for a challenge. If you get a C programmer on the job, he should have little trouble using the Authentication API and calling a standard GSSAPI library to do the actual authentication.

Configuring the custom authentication plug-in is very easy. You can just put the plug-in binary in directory $DM_HOME/dba/auth and it will be loaded by the content server when it starts up.

Tip: If you start the content server executable with option **–o trace_authentication**, it will produce logging of the authentication process in the content server logfile. This is very useful while testing your custom plug-in.

Getting the application to use Kerberos authentication

Now that we have the ability to do Kerberos authentication on the content server, we have just one task left; telling the application to use Kerberos SSO instead of displaying a login screen. Fortunately, the WDK has built-in SSO support that we can leverage.

WDK application configuration

Each WDK application has an **app.xml** configuration file in its <webapp>/custom folder. This file can hold the following SSO configuration parameters:

```
<authentication>
      <docbase>secure_docbase</docbase>
      <service_class>
            com.documentum.web.formext.session.AuthenticationService
      </service_class>
      <sso_config>
            <ecs_plug_in>kerberos5</ecs_plug_in>
            <ticket_cookie>Authorization</ticket_cookie>
            <user_header> REMOTE_USER</user_header>
      </sso_config>
</authentication>
```

<docbase> can be used to restrict the use of SSO to a specific repository

<service_class> should be left as it is

<sso_config> is the interesting bit

- <ecs_plug_in> specifies the Content Server authentication plug-in to use, so here you put the name of your custom Kerberos plug-in
- <ticket_cookie> specifies the name of the browser cookie that will hold the authentication ticket (in our case the SPNEGO service ticket)
- <user_header> specifies the name of the HTTP request header that will hold de user name; this is the header where the app server, or Apache proxy needs to place the user name after it has read and authenticated the SPNEGO service ticket (as explained in the previous chapter)

WDK custom Authentication Scheme

So there you have it. You just specify the plug-in and where to get the username and ticket and your WDK app will start using SSO. Well …almost.

You will have noticed in the specification above that the username is read from a request header and the ticket from a cookie. Unfortunately, when you use Kerberos the ticket is placed in a header called 'Authorization', instead of a cookie.

You can solve this by adding a custom AuthenticationScheme to the application. WDK has a list of authentication schemes that it uses in a set order to try to authenticate users. The properties file **AuthenticationScheme.properties** in WEB-NF/classes/com/documentum/web/formext/session holds the list of authentication scheme Java classes.

You can write your own Authentication scheme class and add it to this list. I used the standard SSOAuthenticationScheme class as a basis for my custom scheme. I only added a few lines of code that copies the 'Authorization' request header into a cookie. The rest of the authorization code remains the same.

Now you're set. SSO authentication between the browser and the application has been arranged, the application has been configured to pass the username and ticket to the content server, and the content server has an authentication plug-in.

Tip: For SSO to work, the username must be the same in the KDC/Active Directory and the Documentum repository. The username is CASE-SENSITIVE (unless you run all servers on Windows), so be careful to only use upper (or only lower) case for the usernames.

DFS

Web services have become more and more popular in Documentum systems in the last few years. Where standard web services can often be called anonymously, this is not true of Documentum web services. For Documentum web services, the same requirements for authentication counts as for Documentum web applications with one exception.

Web services are called by an application, not directly from a user's browser. Therefore, a SSO ticket can never be sent directly from the browser to the web service. It is the application in the middle that has direct contact with the browser, so the application must take care of authentication and send user credentials with the web service request to DFS.

That also means that DFS does not have to be aware of specific details of the SSO solution. DFS supports several ways of authentication:

- If the DFS runtime library is used by the service consumer, it can create a Context and add an Identity object to hold the user's credentials.
- If a regular web service call is made, a WS-Security SOAP Header can be added, holding the user's credentials

In either case, the credentials can be either a username and password (for regular authentication), or a username and SSO ticket. The credentials will just be passed on to the Content Server that will use its authentication plug-in to authenticate the user.

Authentication is transparent for DFS services, only the consumer may need to do some work for SSO authentication.