EMC²
where information lives®

Single Sign-On for Documentum Web Applications with Integrated Windows Authentication

Fangjian Wu

EMC Proven Professional Knowledge Sharing 2010

EMC² PROVEN PROFESSIONAL

Fangjian Wu
Software Architect
A Strategy and Technology Consulting Firm
Documentum.Architect@gmail.com

Table of Contents

Introduction

Have you ever found yourself typing the same user name and password multiple times to access different applications? You may have heard of LDAP, Kerberos, Single Sign-on, and Webtop. Will any of the above help you eliminate the embarrassment of forgetting and typing many Documentum web applications' user name and passwords? Especially when you are required to change your passwords frequently and you can't synchronize them all at once? Would it be possible for you to remember a single password and be done with it? You may be tasked to propose a so-called Single Sign-On (SSO) solution for your Documentum Web applications without adding additional licensing cost. It's difficult, isn't it?

As a Documentum consultant, I have been asked the following questions numerous times by our clients. "Well, why should I enter my Windows login again to gain access to Documentum Webtop, Records Manager, or Web Publisher if it is the same as my Windows workstation login? Or, "This year's budget is really tight because of the recession, I don't have room for additional costs, don't tell me your solution would take weeks to implement" or "How secure your free and open SSO solution would be?" Or "How could Documentum Webtop, a Java-based technology, hey, I know what is Java, possibly be linked with Integrated Windows authentication seamlessly?"

In recent years, we have also seen great technological advances from Microsoft Windows Server, Open Source community, Java Application Server vendors and EMC Documentum. That means something that was unimaginable is now feasible; that translates into a zero cost Windows SSO Integration for Documentum Web applications.

This article examines the concepts and applications of two ticket-based login mechanisms, namely, the Kerberos ticket for Integrated Windows Authentication and the Login ticket for Documentum Content Server. It explores key issues such as Windows IIS and Java Application Server (Tomcat/OC4J) integration, get "Remote User", and LDAP integration. By comparing and linking these two mechanisms, this article offers a straightforward, complete, open, and zero-cost SSO solution for Documentum Web applications. The result is that when a user logs in to their Windows workstation; they are also transparently and automatically authenticated to Documentum Web applications such as Webtop. I will provide two step-by-step reference

implementations with lots of screenshots and diagrams for Webtop running on Apache Tomcat and Oracle Application Server, respectively.

This article only picks the open standard technologies (Kerberos, LDAP) and meets stringent government security standards. It gives the end user community a compelling reason to adopt Documentum Webtop as a single Enterprise Content Management (ECM) platform because it niches out a key differentiator for Documentum. That is, thanks to the Documentum login ticket technology, only Documentum Webtop can be integrated seamlessly with Windows SSO without cost while other “Java-based” ECM products fail.

If you are a Documentum solution architect, Web developer and System administrator, this free SSO solution is for you.

Overview of Single Sign-On Technology

An authentication mechanism is the first defense to secure a computer system or application. As end users moved from dumb terminals to dual monitors in a distributed client/server environment, they inevitably face the challenges of remembering too many User IDs and passwords authenticate. The driving force of any Single Sign-On (SSO) technology is to mitigate this risk.

However, SSO means different things to different people. It could mean LAN-based SSO, Web front-end SSO, or any custom SSO solution. Furthermore, there are implementation issues. The implementations tend to be complex, difficult to integrate and very costly. [1]

This article examines Microsoft Windows SSO and base a Documentum SSO solution on the Windows platform. Windows XP workstation is the de facto standard for corporate users. Most business users use Windows on a daily basis for their work. I can provide the best value by addressing SSO from this (Windows) perspective.

Windows Authentication

Authentication protocol is one of the core security features of the Windows Operating System. Even though most of us use it daily, we barely know how it works behind the scene.

Twenty years ago, in 1990, Microsoft announced LAN Manager 2.0 which was used initially by Windows NT. It can be cracked easily using Rainbow Tables.

Its successor, NTLM (NT LAN Manager), gained widespread use and is well known. It is a challenge-response authentication protocol. It is still being used in some cases to this day, e.g. when the Windows workstation is not in any Active Directory domain (commonly referred to as "Workgroup" or "MSHOME").

Modern Windows systems adopted Kerberos as the preferred authentication protocol. Starting with Windows Vista and Windows Server 2008, both LM and NTLM are deprecated by default. However, we will use Windows Server 2003 as an example in the following discussion since it is widely adopted and installed.

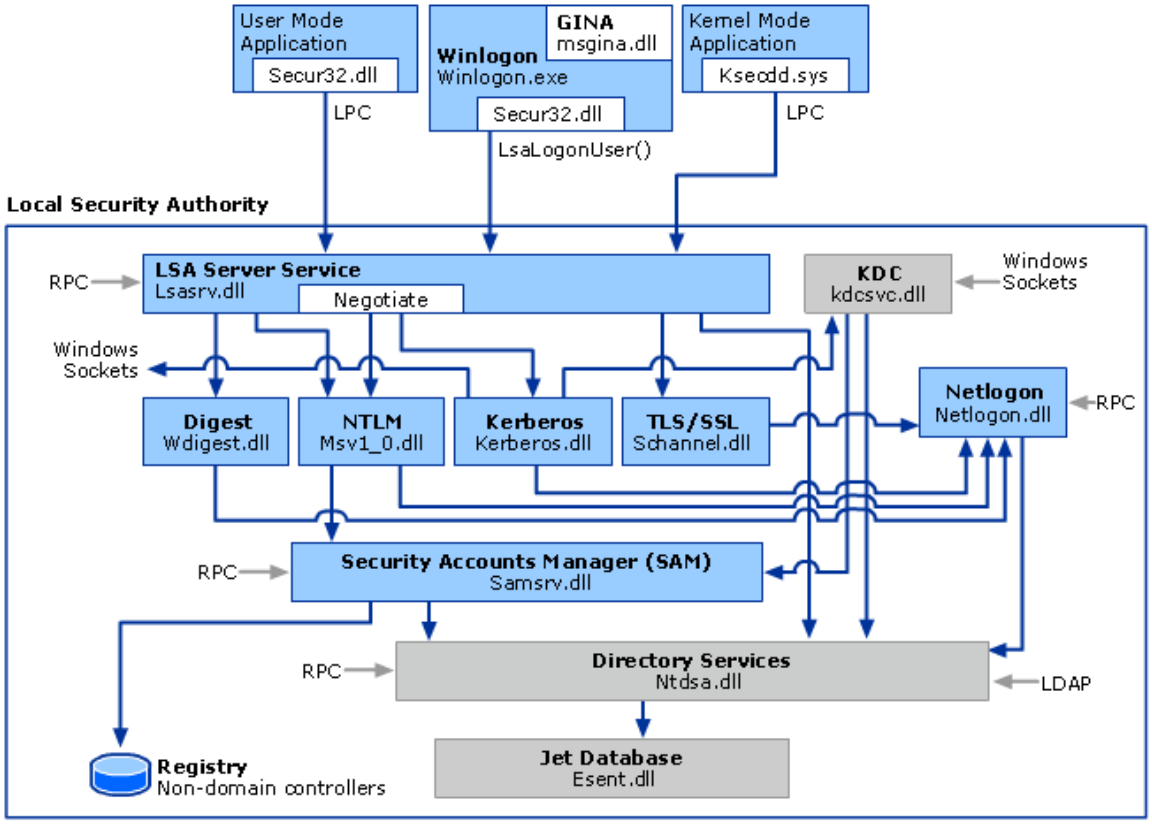
Kerberos is a complex authentication protocol with the following key features: [3]

- An authentication protocol
- Based on encrypted "tickets" with client credentials
- The default authentication package in Microsoft® Windows® 2000
- The basis for transitive domain trusts
- Based on RFC 1510 and draft revisions
- More efficient than NTLM
- Extensible

Kerberos is typically used when a client belongs to a Windows domain. Kerberos builds on the concept of symmetric Needham-Schroeder protocol and requires a "Trusted third party," termed a Key Distribution Center (KDC). Microsoft Windows Domain Controller functions as a KDC. This KDC maintains a database of secret keys; each entity on the network shares a secret key known only to itself and to the KDC.

This transitive trust is the foundation of authentication in Windows Server 2003 and later versions of Windows.

The following diagram provides an overview of all the related Kerberos modules in a Windows system.



(MSDN, Microsoft Corporation)

Single Sign-On

As we all experience in a business Windows environment, users of Windows Server 2003 system don't need to enter a separate credential to access other network file shares. This is because Windows locally caches user credentials in the operating system's Local Security Authority (LSA). When a user logs on to the domain, Windows transparently use the credentials to provide authentication service. This transparency is very neat since it doesn't require extra work for Network administrators.

Authentication API

The logon capability is exposed as a Windows API. For example, in a C# application, we can invoke advapi32.dll trough DLL Import to authenticate user or impersonate a user:
Kerberos Security Service Provider (**SSP**) is also available from Windows.

Windows Integrated Authentication

Other than the client logon API (mainly for Desktop applications), Windows's standard Web Server, Internet Information Services (IIS), offers an "Integrated Windows Authentication" for distributed/web applications.

"Windows Integrated Authentication" uses a cryptographic exchange with the user's Web browser to confirm the identity of the user. Is this cryptographic exchange a Kerberos ticket? Well, it depends on the environment setup.

Authentication Methods

Windows Integrated authentication includes the Negotiate, Kerberos, and NTLM authentication methods. Negotiate is a wrapper to allow the client application to select Kerberos or NTLM for the situation.

For example, if Active Directory is installed on a domain controller running Windows 2000 server or above, and the client browser supports Kerberos v5 authentication protocol, Kerberos v5 authentication is used; otherwise, NTLM authentication will be used. [6]

In other words, if we would like to ensure Kerberos is used, all we need to do is:

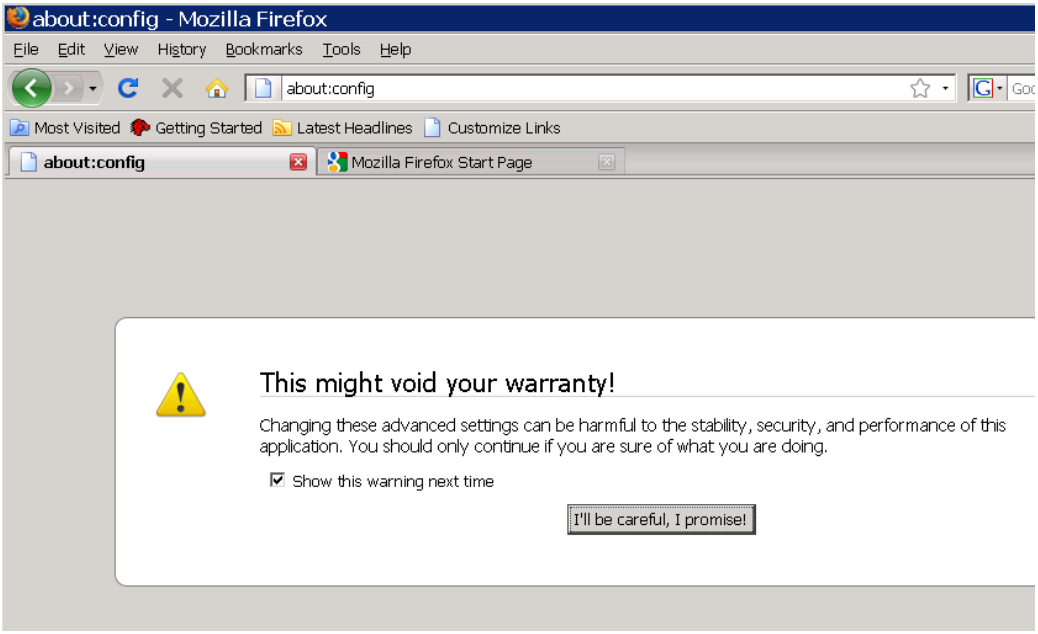
1. On the sever side, Install Active Directory on the domain controller of Windows 2003 (Kerberos by default)
2. On the client side, Use modern Internet Explorer 6.0 or above

As of 2009, most company's intranet environments meet the above conditions with Windows 2003 as Active Directory and Domain Controller and Internet Explorer running at least version 6.0. Therefore, this setup is viable for most of the users in the corporate or government intranet environment.

Firefox

As an end user, we are also interested in asking the other question. Are we limited to a single Web browser such as IE? How about Firefox working with IIS for Windows integrated authentication? Is it possible? The answer is, surprisingly, yes. (I verified it on Firefox 3.0).

1. First, enter "about:config" in the url. When prompted "I will be careful, I promise", click it to confirm



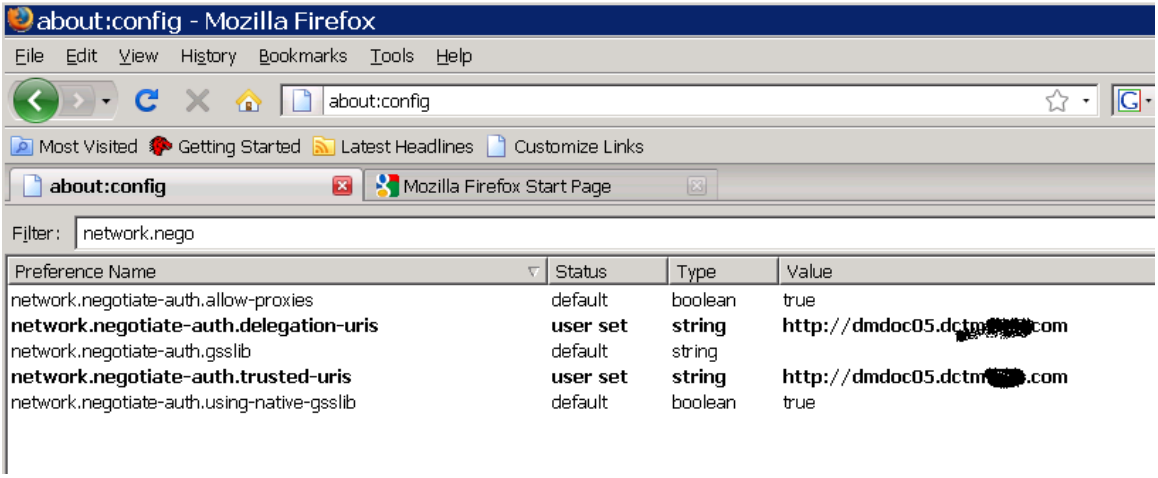
2. In the filter, enter "network.nego"

Locate the following preference names and put the value of the IIS URI:

network.negotiate-auth.delegation-uris

network.negotiate-auth.trusted-uris

Also, ensure that **network.auth.use-sspi** is set to true.



This works because the Firefox code behind the scenes knows the Kerberos and the "Negotiate" protocols.

So far, we have explored some basics of Windows authentication and Kerberos. Let's look at how Documentum Content Server authentication works to further explore how we can benefit from this technology.

Documentum Authentication Mechanisms

EMC Documentum is a full-fledged content management system software suite. It consists of various components such as Content Server, Index Server, Webtop, and many more. The authentication in Documentum is supported in one of several ways:

Default Operation System Account Authentication

The default authentication mechanisms differ depending on whether the operation system is UNIX or Windows. For UNIX, Documentum uses an external password checking program. For Windows, Documentum does not use dm_check_password. Instead, Documentum Content Server authenticates the user's OS name and password within a domain. As a result, you need to set the user's 'user_os_domain' property.

Custom operating System Account authentication

You can create your own password checking program by overriding the 'dm_check_password' program.

LDAP directory server authentication

LDAP is very common in corporate environments. Documentum supports the dm_LDAPSynchronization job to keep Documentum users up to date. This is a single-direction operation. For LDAP users, the property 'user_login_domain' needs to be set to the object name of the LDAP server.

Authentication Plug-in

Out of the box, Documentum supports a few plug-ins for custom authentication. One is for EMC RSA Access Manager, the other one is for Computer Associates (CA)'s SiteMinder. Both plug-ins support Web-based Single Sign-On and strong authentication. The implementation is a platform-dependant DLL library (Windows) or a shared library (UNIX).

In-line password authentication

Documentum can also manage the user name and password separately from any external systems by leveraging an encrypted password that is stored in the 'user_password' property of the user object. [7]

Documentum Login Tickets

Many Documentum users are familiar with the aforementioned authentication mechanisms and have chosen one or more for Documentum application's use. This implies, however, that a user needs to enter his/her user name and password first. Then, these authentication mechanisms will kick in and do their work based on the user name and password entered.

Further research on Documentum session management reveals that there are some other cases that we can securely access Documentum even without a password. For example, there are login tickets, trusted login, and trusted repositories; I will focus on the login tickets here as it relates to the topic of this article.

What is a login ticket?

A login ticket is a token string that you can pass in place of a password to obtain a repository session. You generate a login ticket using methods of an IDfSession object. There are two main use cases.

The first use is to provide additional sessions for a user who already has an authenticated session. This technique is typically employed when a web application that already has a Documentum session needs to build a link to a WDK-based application. This enables a user who is already authenticated to link to the second application without going through an additional login screen. The ticket in this case is typically embedded in a URL.

The second use is to allow a session authenticated for a super user to grant other users access to the repository. This strategy can be used in a workflow method, in which the method has to perform a task on the part of a user without requesting the user password. [9]

We will focus on the second use for SSO authentication in this paper.

DFC

First, create an IDfSessionManager and an IDfSession with an admin account to implement the login ticket in DFC. Then, use this adminSession to generate Tickets for any specific user.

IDfSessionManager sessionMgr = this.createSessionManager(docBaseName, **AdminUser**, **AdminPassword**, "");

IDfSession **dfSession** = sessionMgr.getSession(docBaseName);

String **myTicket** = dfSession.getLoginTicketForUser("SpecificUser").toString();

Once you obtain the ticket, creating a normal user session using DFC is as simple as:

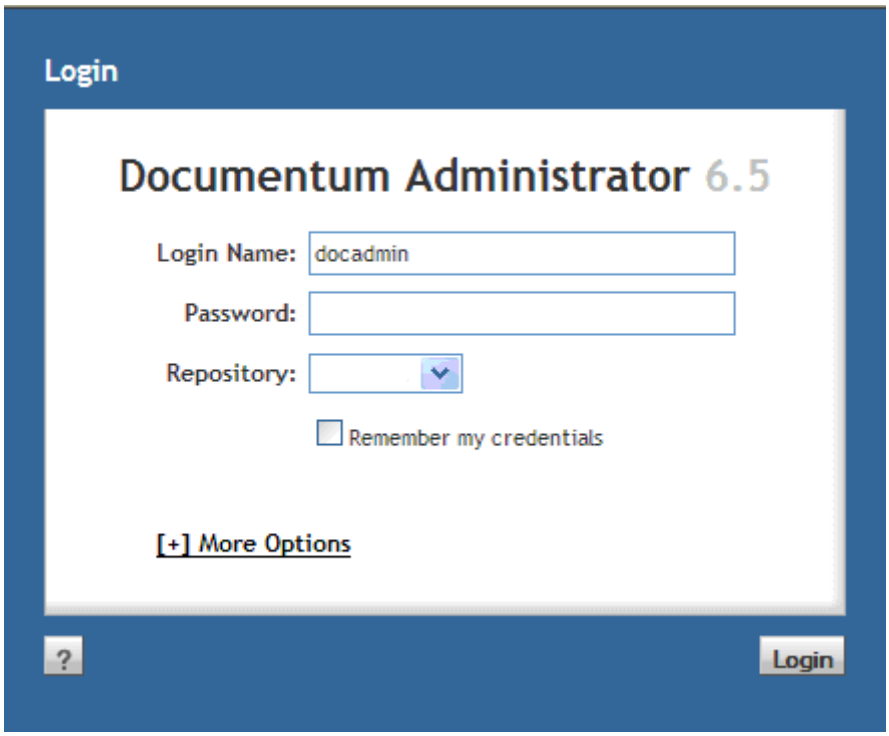
```
DfClientX clientx = new DfClientX();
IDfLoginInfo loginInfo = clientx.getLoginInfo();
loginInfo.setUser(userName);
```

```
loginInfo.setPassword(myTicket);
// if an identity for this repository already exists, replace it silently
if (userSessionMgr.hasIdentity(repository))
{
userSessionMgr.clearIdentity(repository);
}
userSessionMgr.setIdentity(repository, loginInfo);
// get the session and return it
IDfSession session = userSessionMgr.getSession(repository);
```

To troubleshoot a login ticket issue, use IDfSession.getLoginTicketDiagnostics, As you can see, it's fairly straightforward in DFC, but how about in WDK/Webtop?

WDK/Webtop Application

Webtop is a Documentum Web application built on top of WDK framework which is built on top of the J2EE platform. All WDK applications share a similar login Windows as follows:



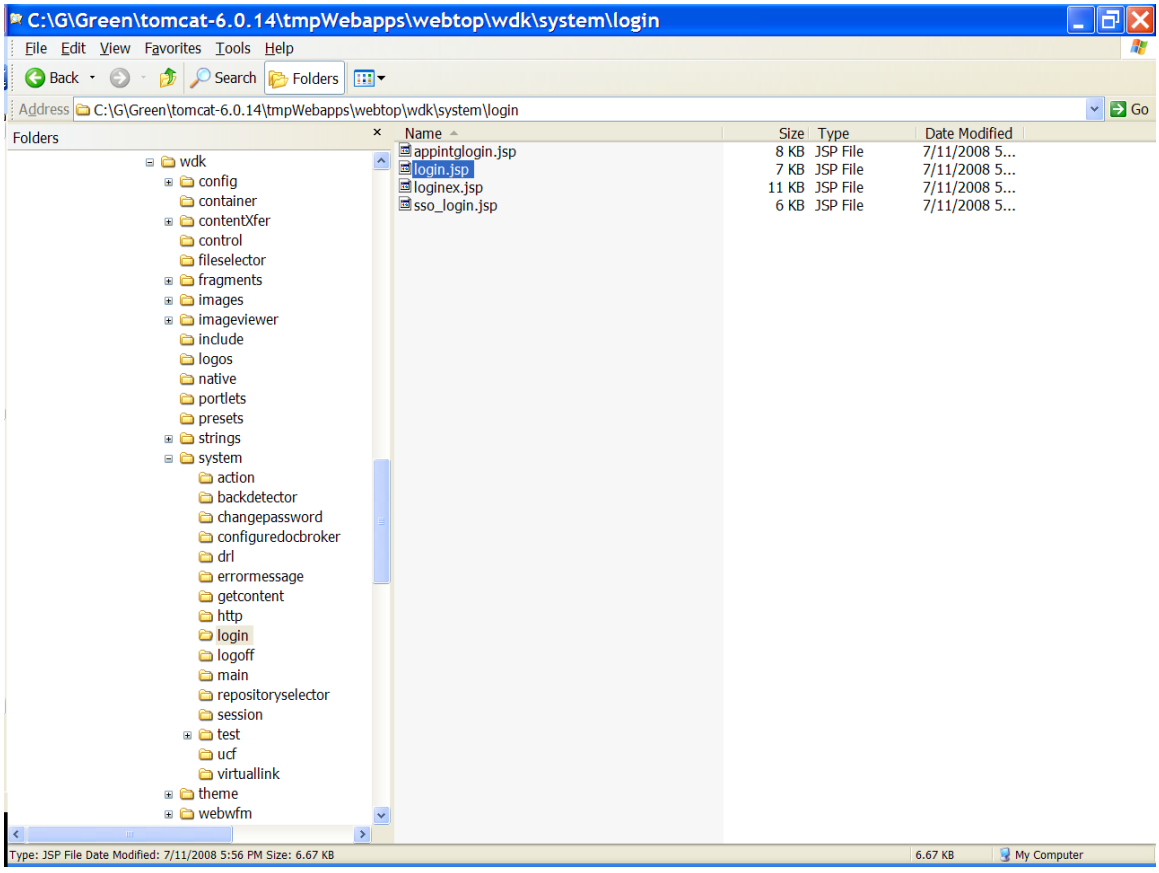
This is a login.jsp Window that uses the credentials entered by the user to generate a Documentum session.

Customizing login.jsp

The login.jsp is customizable based on your needs. It is located at the following URL:

[WEBTOP_HOME] \wdk\system\login

WEBTOP_HOME is the top level webtop folder. Usually named “webtop”, it should have a direct sub-folder called “WEB-INF”.



It is possible to customize this page in multiple ways. I am only providing one possible solution in this article.

First, we make sure that in the JSP, a Java HttpRequest object call, **request.getRemoteUser ()**, returns a valid user name. Then, this user name can be used to generate a login ticket. **SSOLoginTicket** is a DFC wrapper class that creates the adminSession and generates the login ticket as shown in our DFC example.

```
…
<%
        String loginTicket = "";
        String userName = "";
        SSOLoginTicket ssoLoginTicket = new SSOLoginTicket();

        if(request.getRemoteUser()!=null) {
                userName = request.getRemoteUser();
        }
        userName = userName.substring(4);

        System.out.println("!!! Final User Name [" + userName+"]");

        loginTicket =  ssoLoginTicket.generateLoginTicket(userName);
%>
…
```

On the same login JSP page, an auto-loaded JavaScript function init() will be placed after the HTML forms code to use this “loginTicket” to replace the value of the LoginPassword HTML box and submit the form by firing off a button click() event.

```
…
<script>
function init() {
        var loginTx;
        var loginUserName = document.getElementById("LoginUsername");
        var loginPassword = document.getElementById("LoginPassword");
        var loginButton  = document.getElementById("LoginButton");
        loginUserName.style.visibility="hidden";
        loginPassword.style.visibility="hidden";
        loginButton.style.visibility="hidden";
        document.getElementById("DocbaseName").style.visibility="hidden";
```

```
        var loginTx = '<%=loginTicket%>';
        var spans;
        if(document.all) {
                spans = document.all.tags('span');
        }
        else if (document.getElementsByTagName) {
                spans = document.getElementsByTagName('span');
        }
        if(spans.length < 1) {
                if(loginTx.length > 0) {
                loginUserName.value = '<%=userName%>';
                loginPassword.value = '<%=loginTicket%>';
                loginButton.click();
                }
        }
}
window.onload=init;
</script>
…
```

In this case, a user doesn't need to do anything. If request.getRemoteUser() doesn't return a valid login user name, the login will simply fail. When in a Windows domain, the getRemoteUser() will return in a format of "Domain Name\User Name".

userName.substring(4) here is to strip the domain name off the user name. For example, if my test domain name is "SUN" so that the initial username would be "SUN\John.Smith". Then, substring(4) will return only "John.Smith". You would need to tweak it depending on your target domain name. For example, you can use IndexOf ('\') function to get the index, then use the substring method.

Obviously, the valid user name needs to match an existing user account in the Documentum system. Otherwise, a user name is valid in the Windows domain but not in Documentum. For synchronization purposes, as a Documentum administrator, you can run the LDAP synchronization job or create your own job to add the valid users to Documentum. The

illustration here is for a single DocBase. For multiple DocBases/Repositories, the Dobase/Repository name can also be passed into the JSP via Http URL query string.

One thing worth noting is that the AdminPassword is required to be passed in to **SSOLoginTicket**. Therefore, distributing this password in clear text could be a problem.

To avoid this, you can use multiple ways to encrypt and decrypt this admin password. For example, you can Google and find a standard Java Triple DES implementation to suit your needs. In WDK, there is built-in support for encrypting a password with the trusted authenticator tool (**com.documentum.web.formext.session.TrustedAuthenicatorTool**). This tool uses stores the symmetric key in a file named “wdk.keystore”. You must put this file in a secure and protected location.

By now, we know as long as request.getRemoteUser() is retrieved from a JSP page, everything else is workable. But how do we get a remote user?

We remember that a Windows Integrated Authentication IIS application can have a simple asp page (**user.asp**) that returns the RemoteUser server variable easily.

```
<HTML>
<BODY>
<TABLE>
        <% For Each name In Request.ServerVariables %>
    <TR>
        <TD>
            <%= name %>
        </TD>
        <TD>
            <%= Request.ServerVariables(name) %>
        </TD>
    </TR>
    <% Next %>
</TABLE>
```

</BODY>
</HTML>

Unfortunately, this is not a JSP page. But there might be some ways to integrate them. First, let's look into J2EE Server Component Model closely to see they can be integrated.

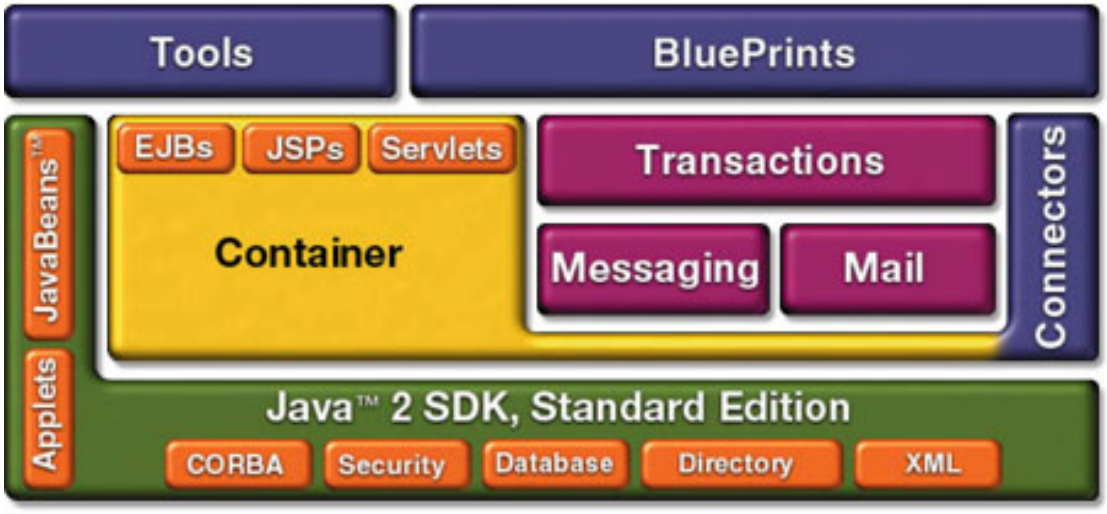
J2EE Server Component Model

Starting from version 6, Documentum Webtop is designed to be deployed to a J2EE Application Server with a J2EE war file (webtop.war). The same war file can be deployed to any J2EE-compliant application server. For example, Apache Tomcat 6, Oracle Application Server 10g, and Glassfish are all good choices (J2EE licensed and Certified).

For specific features, we can refer to theServerSide.com's Application Server Matrix at (http://www.theserverside.com/tt/articles/article.tss?l=ServerMatrix)

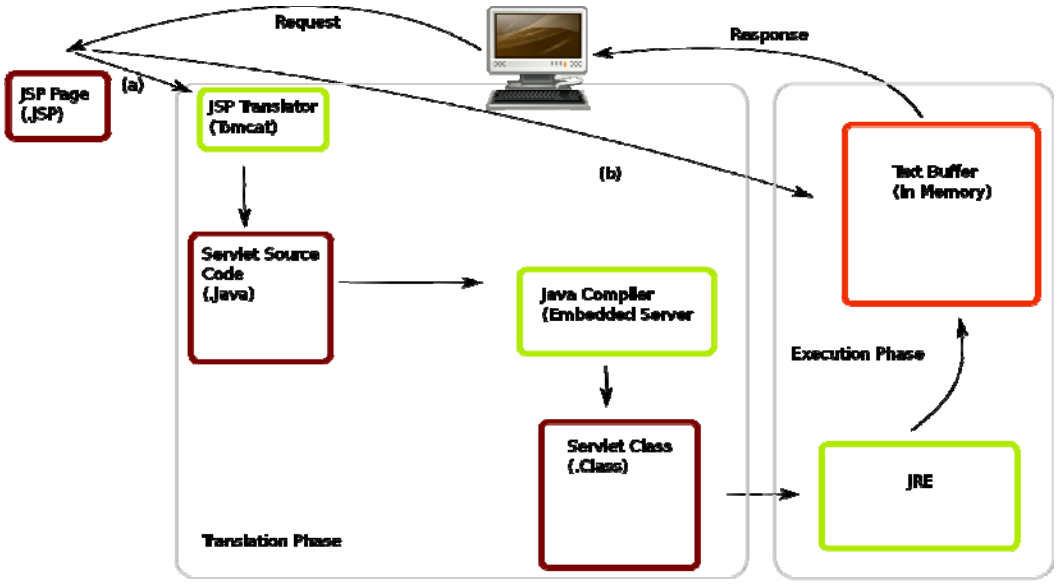
JSP Model

Regardless of their J2EE implementation, the core component model, JSP/Servlet container, is identical by the specification.



(www.java.com)

As a core component of J2EE, a JSP container is essentially a page compiler that can compile a JSP script into a Java Servlet class. The Servlet class then runs within the same container JRE and serves the requests submitted to it as illustrated in the following diagram:



JSP Container
(a) Translation occurs at this point if JSP has been changed or is new
(b) if not, translation is skipped.

(Wikipedia)

An integrated HTTP Listener (A.K.A Web Server) is one of the key features implied in a modern JSP container. This front Web Server is supposed to accept the initial HTTP request and serve the static HTML content to the Web Browser. If the request is a JSP page, it will route it to the JSP container and forward the response from JSP container back to the Web Browser. What this means is that a HTTP listener is not necessarily bound to a JSP container. It can be a separate external entity. You can specify which external web server to use in the old JRun (a popular J2EE server in the market around 2000-2002), in the installer.

The JSP container was designed to serve the dynamic content, not as a general-purpose Web Server. It's very common in a production deployment that uses an external front-end Web Server rather than the built-in one. The external full-fledged Web Server offers much more powerful features (for example, load balancing) and performance gains.

Apache Web Server and Microsoft IIS are very popular for production deployments.

AJP13

But how does Apache or IIS integrates to a JSP container? Unfortunately, there are no defined standards. We must evaluate on a case by case basis.

I have seen that some JSP containers will involve a less-known protocol called AJP13 (Apache JServer Protocol, version 1.3). This means the Java Application Server side will listen to an AJP protocol port.

To facilitate the use of their Java Application Server as the middle-tier server for their customers, most of J2EE server vendors offer a free plug-in (usually written in C and release as a shared library) to a specific Web Server, in this case, Apache or IIS. In this article, I will provide working AJP13 examples for Apache Tomcat JSP container and Oracle Application Server 10g.

The reason why I am choosing these two is that they are representative and can be suitable for your project needs.

Oracle Application Server 10g Container for J2EE, A.K.A "OC4J", is usually a commercial licensed product bundled within other enterprise-level licenses. The cost model is up to your organization's licensing agreement with Oracle. It could be expensive. I have seen many government agencies committed to this J2EE sever and moving in the direction of standardizing on it.

Not surprisingly, OC4J offers support for integrating with external Web Servers and the documentation on AJP13 is generally good. You get what you pay for.

Apache Tomcat 6, on the other hand, is open source and free (Apache-licensed). It is a good choice for budget-sensitive organizations. It lacks some extra enterprise-level features; however it is very stable, J2EE-certified, and well-supported by good documentation, numerous user forums, and third-party service providers. Sometimes, the disadvantages of fewer administrative modules and J2EE features (e.g. no EJBs) might be an advantage. It means less start-up time, better performance, fewer constraints on system requirements. The AJP13 support documentation is also good. Did I mention AJP13's "A" means Apache here? In my experiment, I got this implementation working before I could get O4J's implementation working.

This is the basic setup; the added security layer is possible. For example, we can setup a Linux or Windows Firewall on the Java Application server side to only allow certain IP addresses coming through the AJP ports to enhance security if required.

Solution Design - Putting things together

A technical solution is designed to address a typical business problem based on some assumptions within technical boundaries. This article is no exception. We will come up with a common use case first. This use case needs to be targeted to suit the needs for the majority of you who are reading this article.

The Business Use Case:

Acme Company is a publishing business with 1000 Documentum users. For an enterprise-wide Documentum implementation, Acme has the following software and hardware infrastructure:

1. A Server running Windows Server 2003 is the Active Directory/Domain Controller for the domain “ACME”.
2. All the “ACME” domain users are using a standard Windows XP workstation within a firewall
3. The standard Web Browser is IE 7
4. EMC VMWare Virtual Infrastructure Hosting Documentum System
    a. Documentum Content Server on Linux (RedHat) Server
    b. Documentum Webtop running on a Windows Server
    c. The Application Servers supporting Webtop is OC4J.
5. The stakeholder has already secured all the above licenses.

The Task:

The end user community complains that once a user logs in to his Windows XP machine, when he uses the Internet Explorer web browser to access Documentum Webtop URL, he is still being prompted by a login window.

As a Documentum manager/architect, you are tasked to implement a Documentum-enabled SSO solution for the Documentum users.

Due to the budget constraints, however, the cost should be minimal and the deadline is yesterday.

The Assumptions:

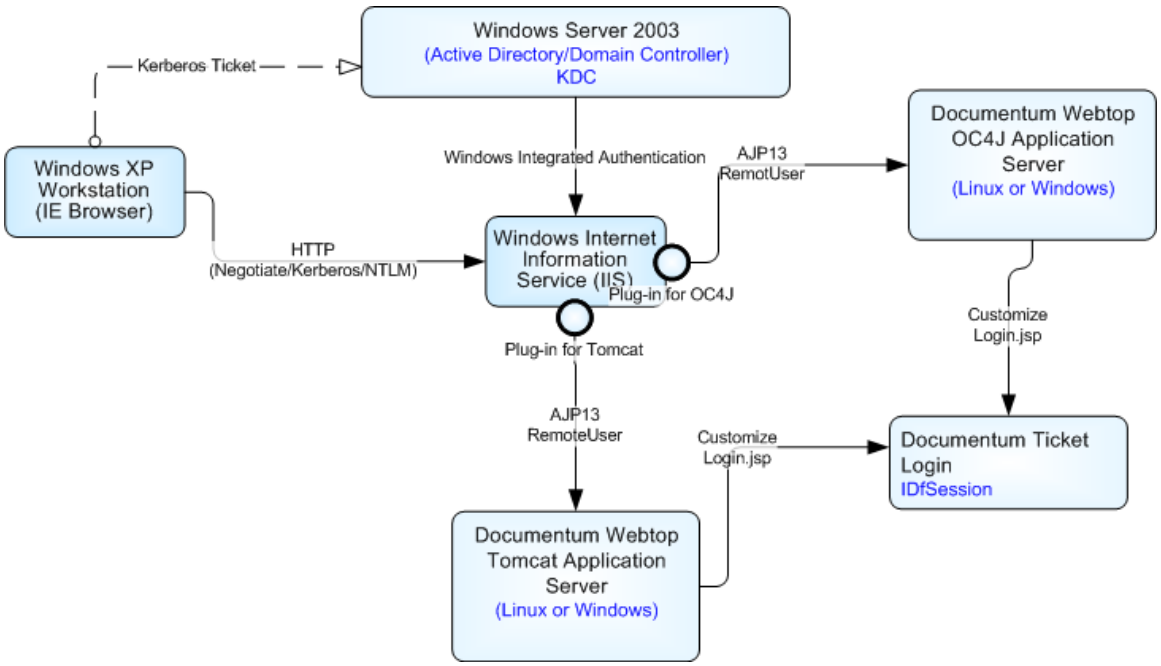
We assume Acme has existing support contracts with Microsoft, Oracle, and Documentum in place. Provisioning an IIS front-end Web Server means no extra cost.

The Goals:

1. We will implement a SSO solution without additional software or hardware licensing cost
2. The labor cost (a couple of hours maximum) preferably is close to **zero** except the time you spend following instructions in this article.
3. It is transparent to the existing system so that if in the future, you decide to adopt another paid SSO solution, you can easily swap out this solution within minutes.
4. This solution will reply on the secure Integrated Windows Kerberos technology. Not only does it tie to the Windows XP workstation login to eliminate the extra Web SSO layer, but also guarantees that ongoing security will be well protected, and if needed, patched by Microsoft in the long run.

The Steps:

With a good understanding of how Kerberos ticket works for Windows, how Documentum login ticket works and how J2EE Server works, we are confident that we can put all the pieces of this puzzle together as illustrated in the following diagram.



We know we can leverage Windows authentication for Desktop and Web application from our analysis of Windows authentication and Kerberos ticket. We also know that the Internet Explorer and IIS Web Server will support Kerberos when conditions are met.

Our steps are:

1. Setup IIS Windows Integrated Authentication

   This front-server is to ensure Kerberos ticket and authentication work.

2. Test user.asp to make sure ASP remote user is valid

   If ASP can return remote user, we know the HTTP request remote user can be used to identify if a user has passed Windows integrated authentication.

3. Setup IIS plug-in for the J2EE application

   The plug-ins need to setup and configured to redirect the HTTP request along with the authenticated remote user information to the Java Application Server

4. Setup Java Application Server to support AJP13

   On the Java Application Server side, we need to change some configuration information to enable AJP13 listener.

5. Test test.jsp to make sure JSP remote user is valid

   Once AJP13 is enabled, test to see if the HTTP request still works in terms of Remote User server variable for JSP.

6. Customize Webtop login.jsp

   After JSP passes the correct remote user, customize the Webtop to initiate the Documentum ticket login process.

As can be seen fro the diagram, our design is very cost-effective and technologically compelling. Rather than requiring a big dollar investment and months of implementation, this solution doesn't need extra licenses and requires minimal labor. The IIS is a built-in component of the Windows Operating System and doesn't require separate licenses. The Active Directory/Domain Controller is usually given when you login to your workstation. The IIS Plug-ins for a specific application server are generally free and supported by the Application Server vendor. Best of all, due to the modern Kerberos technology, it is secure, transparent, and fully supported by Microsoft as part of the existing Windows licenses.

There are no hidden or esoteric technologies involved. Everything we followed here is open and standard technology. However, we are not tinkering on our own. We are standing on the shoulders of the giants such as Microsoft, Oracle and EMC. If we can trust Windows operating system as our daily workstation, why don't we let it do the SSO job that Windows has already offered? Another potential benefit is Documentum seamless SSO integration with Microsoft SharePoint System based on the same technology. We can come up with another business use case for that.

The only code update (one single JSP file) on the Documentum side leverages a very good feature in the Content Server, Ticket login, as well as the Webtop's customizable nature. Since AJP13 is a cross-platform protocol, our Application Server could be running on any platform (Unix/Linux/Windows). We are taking advantage of all the best possible capabilities in a modern corporate environment.

The following two implementations will illustrate the above steps by showing real examples targeting two popular Application Servers that Documentum has long time supported.

Implementations

Case Study 1 – Apache Tomcat 6.0.14

Tomcat 6 is a straightforward JSP/Servlet Container. It is one of the popular Application Servers that Documentum supported since Documentum WDK 5. We can follow the design from the last section and implement the SSO step by step.

Before we get started, we need to download the software component.

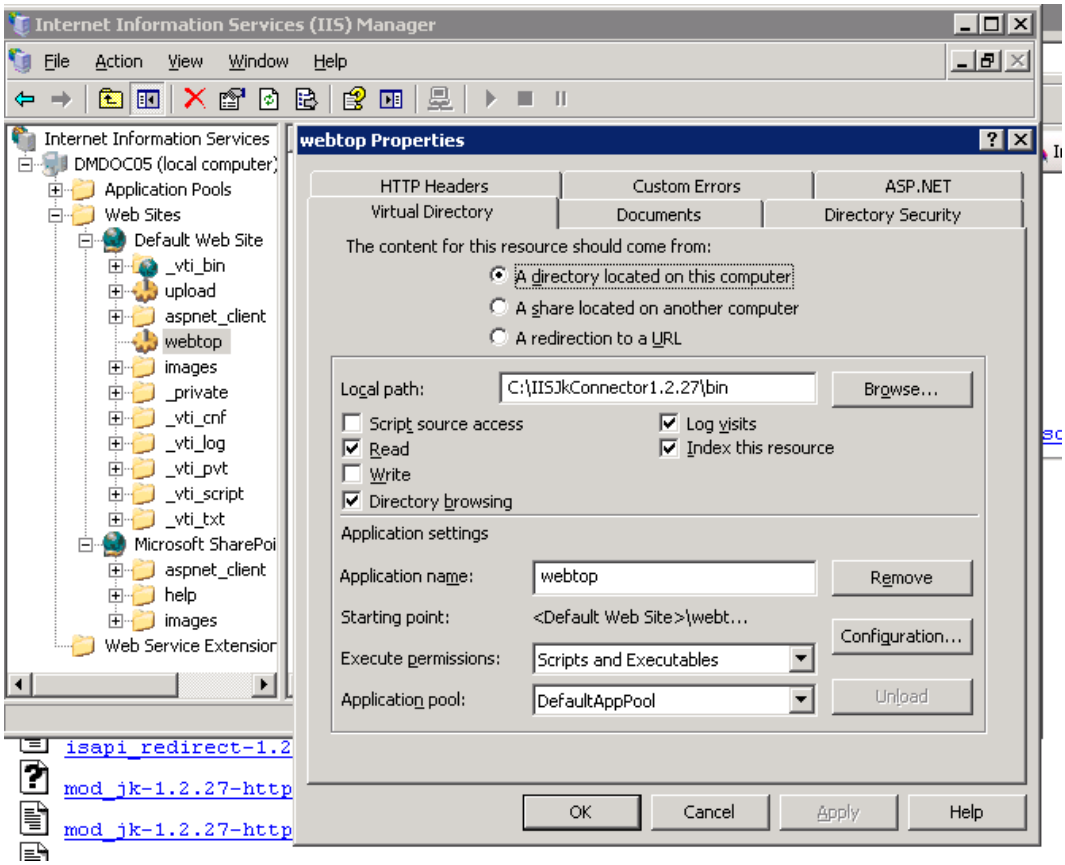
- IIS Plugin for Tomcat (isapi_redirect-1.2.27.dll)
  http://archive.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/win32/jk-1.2.27/

Steps:

1. Setup IIS Windows Integrated Authentication

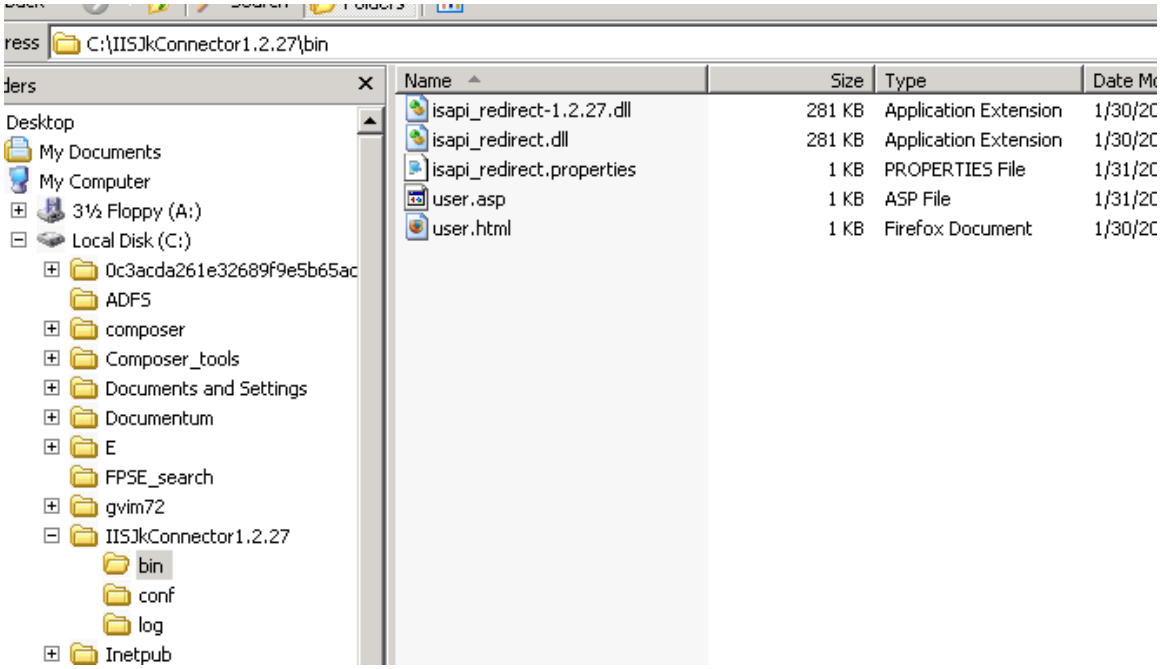
a) Create a Windows virtual directory called “webtop”, point it to a local folder C:\IISJkConnector1.2.27\bin

Illustration is on the following page.



Next, put **isapi_redirect-1.2.27.dll** in this folder, rename it to isapi_redirect.dll and create a text file, **isapi_redirect.properties,** in the same folder.

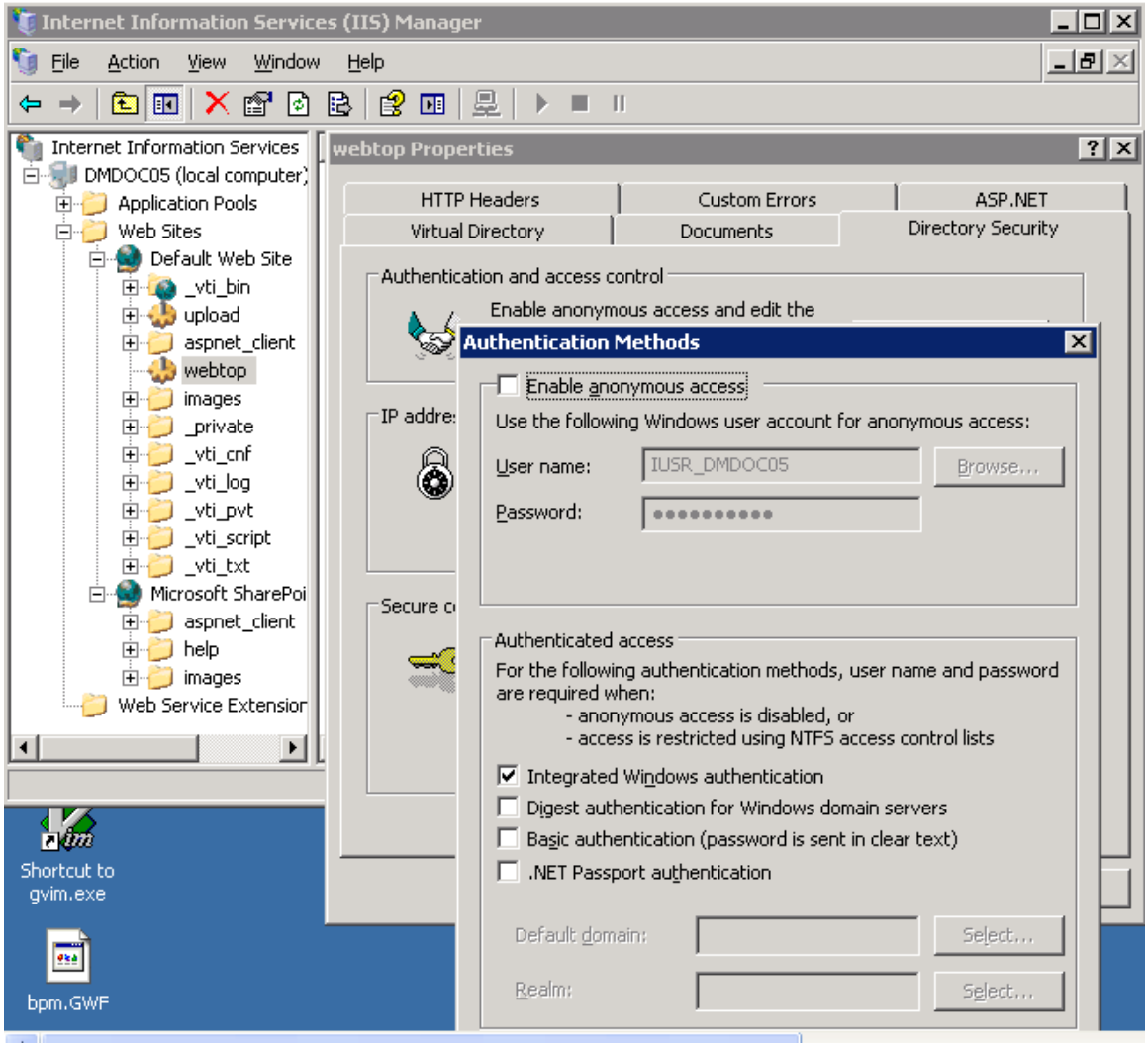
Also, create test files, user.html and user.asp in this folder



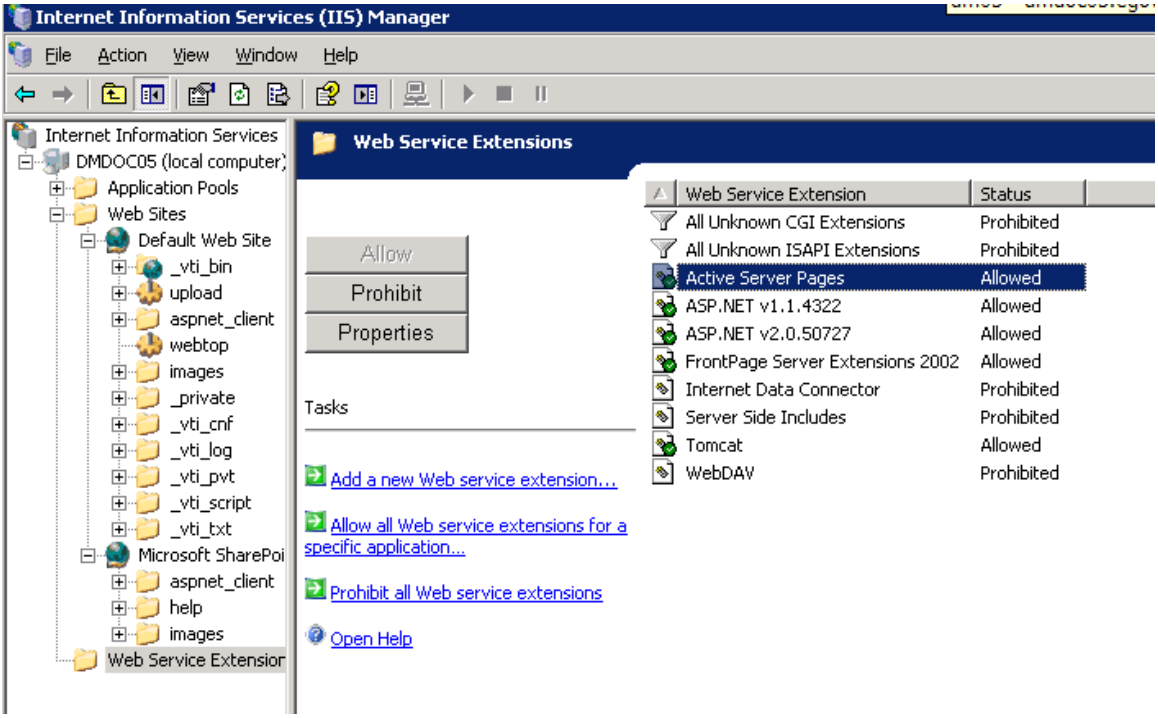
The content of user.html could be anything but **user.asp**. You can also pick only the “Remote user” server variable as follows:

<%= Request.ServerVariables(“REMOTE_USER”) %>

Next, go to properties of “webtop”, select tab “Directory Security”, then, click the “edit” button of “Authentication and access control”. Once the Authentication Methods window launches, makes sure “Integrated Windows Authentication” is checked and “Enable Anonymous access” is unchecked.



Last, make sure Active Server Pages is allowed from “Web Service Extension”



Once the IIS web application “webtop” has been setup, we can move to next step, “Test user.asp”.

2. Test user.asp to make sure ASP remote user is valid

First, open your IE browser and test the static HTML by entering

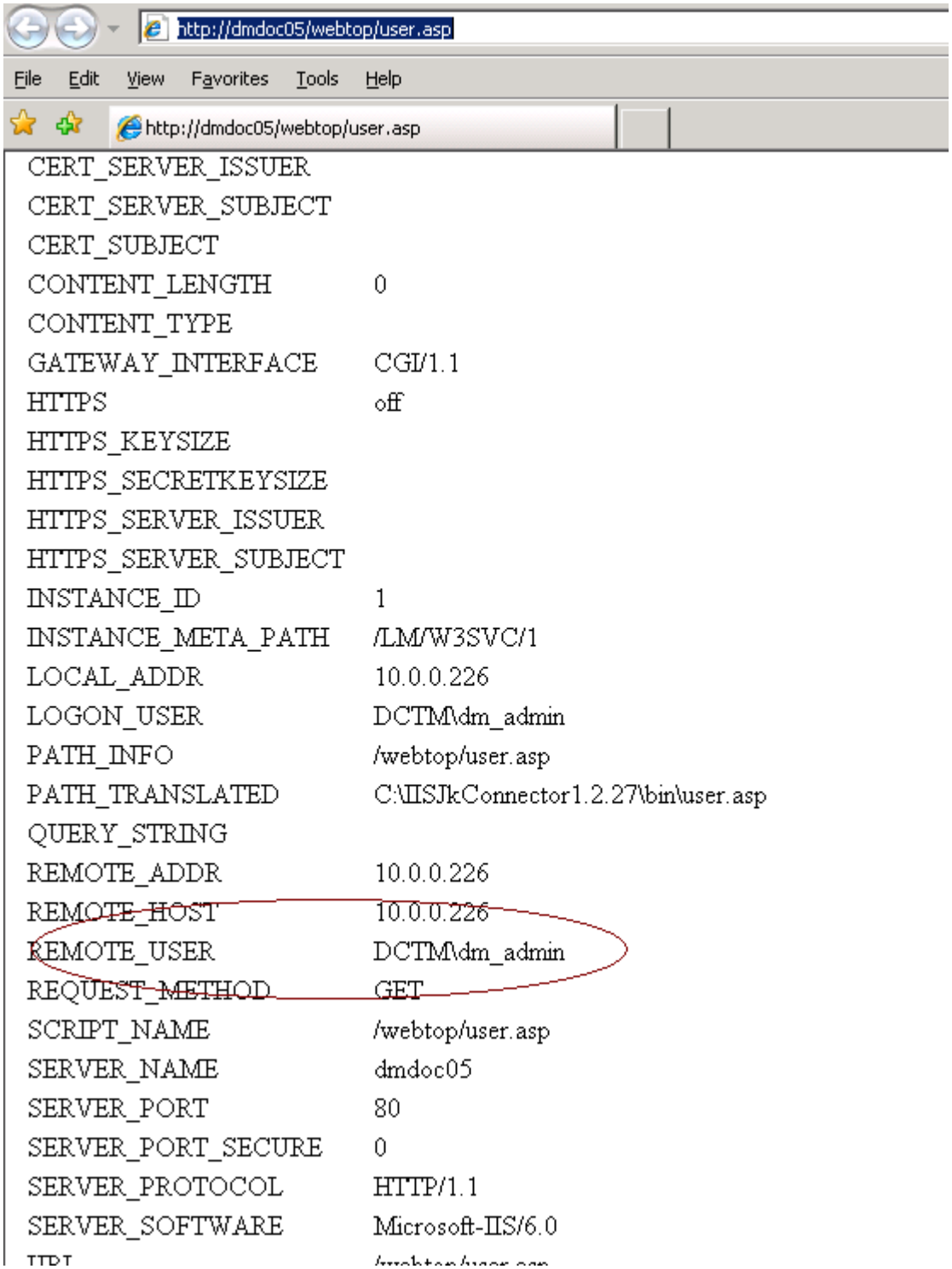
http://hostname/webtop/user.html

If it is good, move to next step Otherwise, please consult with your Network Administrator to check if IIS has been installed correctly.

Now, we can verify if ASP works by entering

http://hostname/webtop/user.asp

You should see a page displaying all server variables:

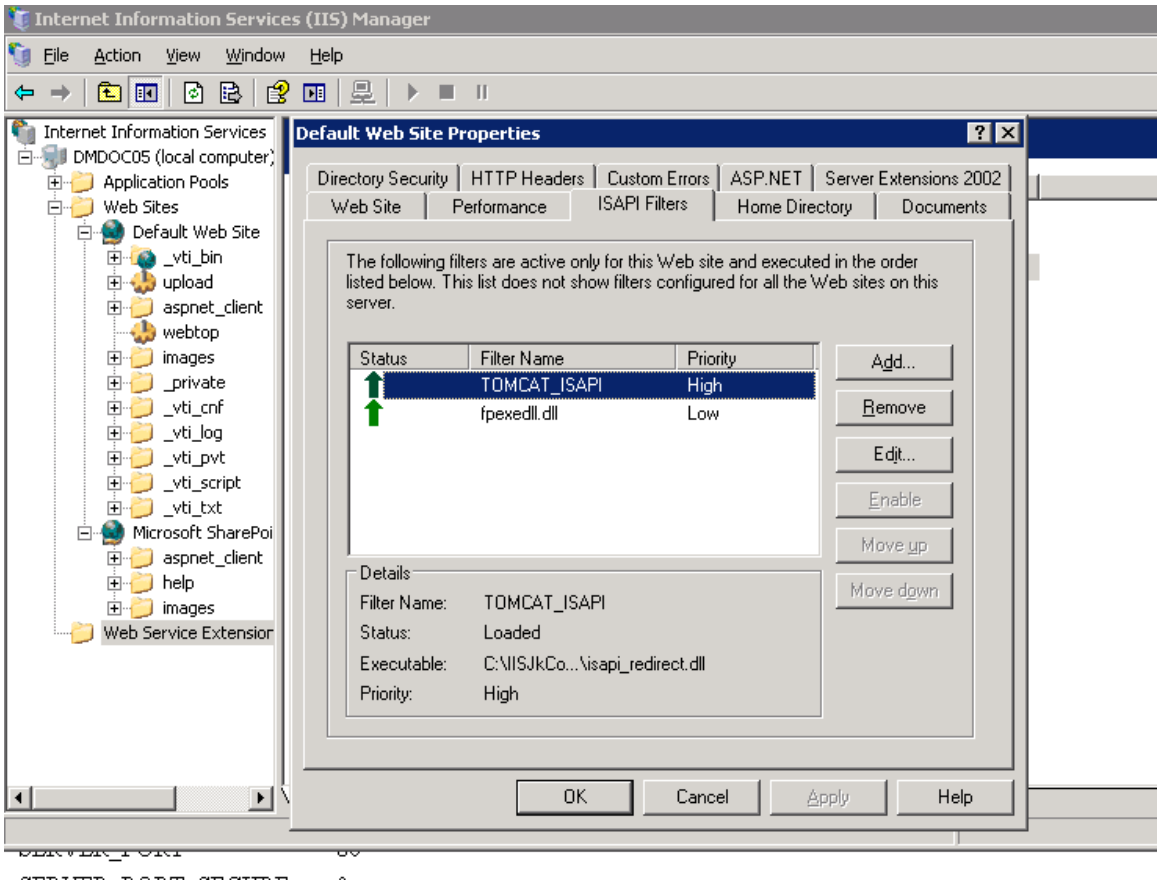


Please notice "Remote_User" is set and valid. By now, we know the “webtop” ASP application works with Windows Integrated Authentication.

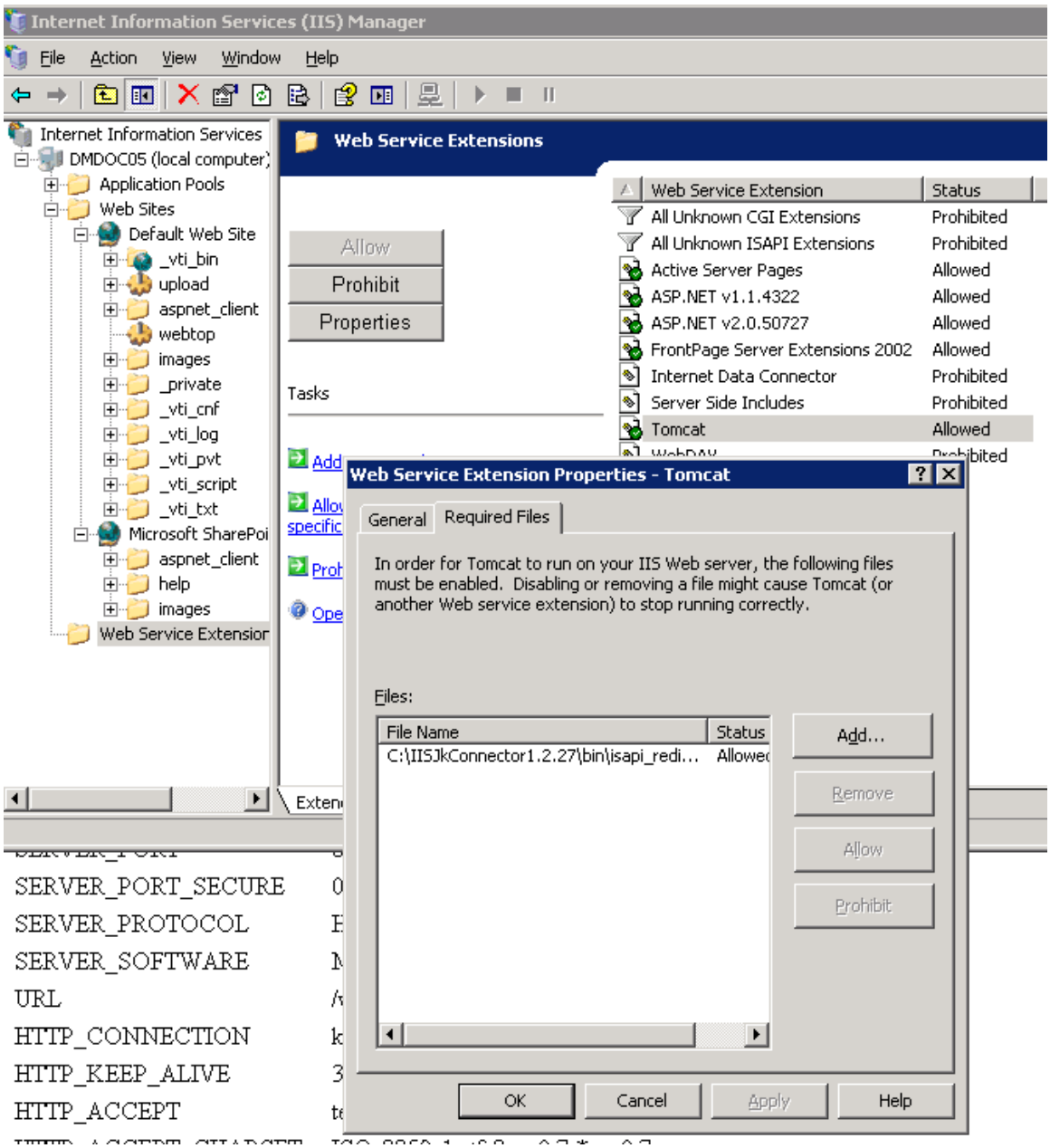
3. Setup IIS Plug-in for Tomcat J2EE applications

To set up IIS Plug-in, first, we need to setup an ISAPI Filter so that the Plugin DLL will be loaded and start filtering HTTP requests when the IIS service starts up.

Go to the properties of “webtop” and select tab “ISAPI Filter”, then add TOMCAT_ISAPI and point it to **C:\IISJkConnector1.2.27\bin\isapi_redirect.dll**



Next, make sure this Tomcat DLL is allowed from the “Web Service Extension” similar to what we did to “Active Server Pages” to enable ASP.



Next, configure the Plug-in itself. The first file is **isapi_redirect.properties** we just created in step 1. We will add the follow content to this file.

extension_uri=/webtop/isapi_redirect.dll
log_file=C:\IISJkConnector1.2.27\isapi_redirect.log
log_level=info
worker_file=C:\IISJkConnector1.2.27\conf\worker.properties
worker_mount_file=C:\IISJkConnector1.2.27\conf\uriworkermap.properties

Then, go to C:\IISJkConnector1.2.27\conf and create two text files. Name them **worker.properties** and **uriworkermap.properties**

For **worker.properties**, we add the following lines:
worker.list = my-tomcat-worker
worker.my-tomcat-worker.type = **ajp13**
#Java Tomcat host name
worker.my-tomcat-worker.host = **DMDOC04**
worker.my-tomcat-worker.port = **8009**

For **uriworkermap.properties**, we add the following lines:
/webtop/* = my-tomcat-worker

The above host (**DMDOC04)** and port (**8009)** are Tomcat hostname and AJP13 ports. We will get more into that in next step. The IIS and Tomcat may or may not be on the same physical machine.

After you have done all of the above, restart the IIS services from Windows Control Panel Administrative Services.

4. Setup Tomcat to verify AJP13 and ports

It is very easy to set up Tomcat. We only need to open up the AJP13 protocol so that IIS's AJP13 redirection request won't be rejected by Tomcat. In other words, Tomcat's AJP13 listener should be up and listening to port 8009 that we just configured above.

Just go to **server.xml** which is located in **C:\tomcat-6.0.14\conf**, and make sure the following is not commented out:

```
  <!-- Define an AJP 1.3 Connector on port 8009 -->
    <Connector port="8009" propertiesFile="conf/jk.properties" enableLookups="false"
protocol="AJP/1.3" redirectPort="8443" />
```

My **jk.properties** file has the following two lines:
request.tomcatAuthentication=false
request.registerRequests=false

Now, it's a time to startup the Tomcat instance.

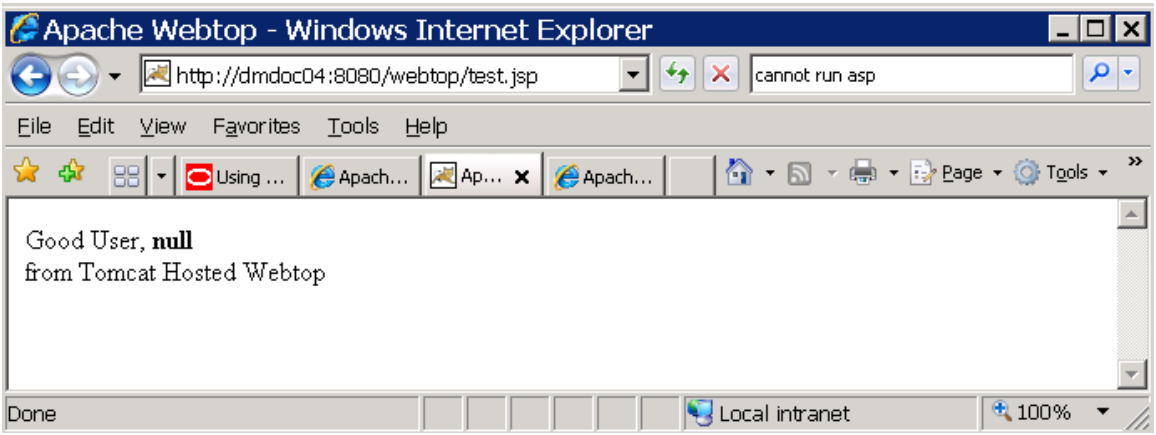
5. Test test.jsp to make sure JSP remote user is valid

This is a critical step in that we expect the plug-in will function correctly.

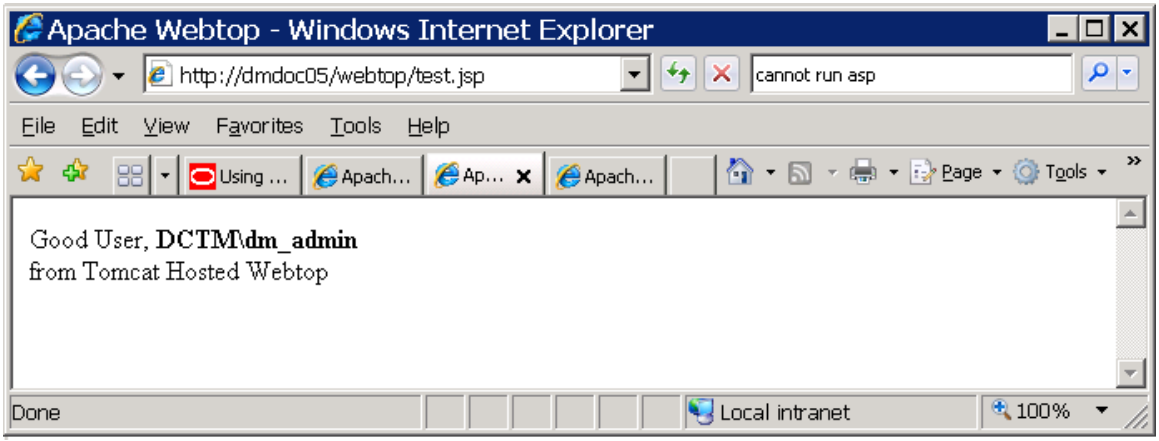
Creating a little double "webtop" J2EE application is any easy way to do it. It is a tiny JSP Web application called "webtop" and contains only one JSP file, **test.jsp**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page contentType="text/html;charset=windows-1252"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"/>
    <title>Apache Webtop</title>
  </head>
 <body>Good User,  <b> <%=request.getRemoteUser()%> </b><br> from Tomcat Hosted
Webtop</body>
</html>
```

Once this little webtop.war file is deployed to Tomcat, if we hit Tomcat directory (**DMDOC04**) (hitting Tomcat's built-in Web server), we get:



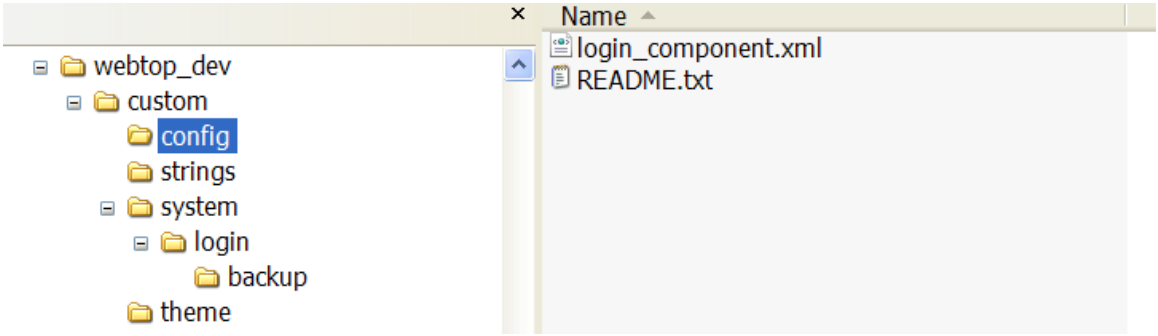
However, if we enter the URL pointing to the IIS server (**DMDOC05**) first, which redirects to DMDOC04 (Tomcat) after Windows integrated authentication, we get:



This is the result that we expected from the test. It means the HTTP request first goes through the Windows Integrated Authentication, then routes to the Java JSP with the correct Remote User in the request. If I try to bypass the IIS and Windows integrated authentication, the remote user will always be **null**.

6. Configure Customized Webtop login.jsp

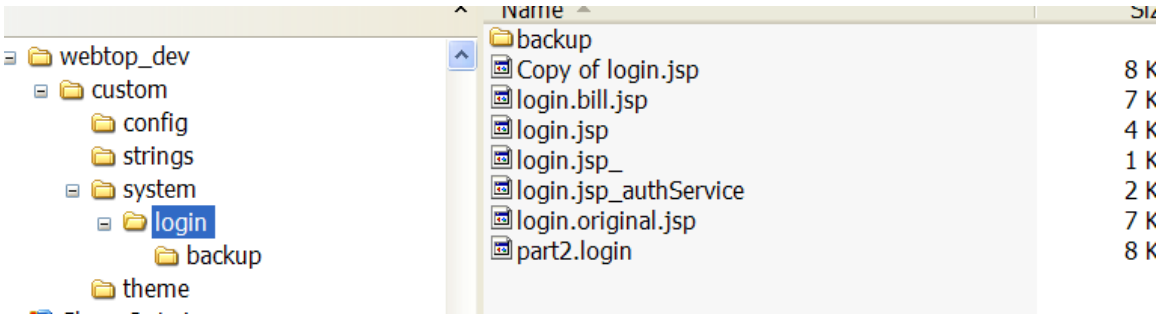
At this point, a Documentum webtop developer should be familiar with the process. We start by adding a "/**custom/config**" folder under the Webtop root and add **login_component.xml**



This file should contain a customized <component> tag such as:

```
<component id="login" extends="login:webtop/config/login_component.xml">
        <desc>
          Extends the WDK login component and displays a login page, adding
          a Webtop application identification string in the login screen.
          The login component is called by the component dispatcher if a
          component is called without a valid session.
        </desc>
                  <pages>
            <start>/custom/system/login/login.jsp</start>
        </pages>
        <nlsbundle>com.documentum.webtop.session.LoginNlsProp</nlsbundle>
        <centered>true</centered>
      </component>
```

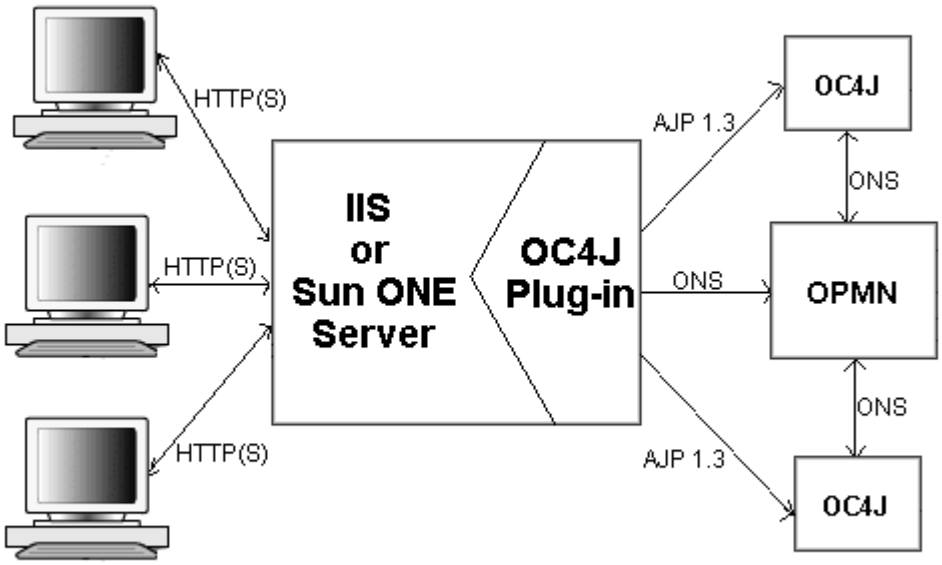
Then, place your customized **login.jsp** under **/custom/system/login:**



As in any J2EE application, place the supporting wrapper class, **SSOLoginTicket.class**, to the **WEB-INF\classes** folder or bundle it into a Jar file and place it to **WEB-INF\lib**.

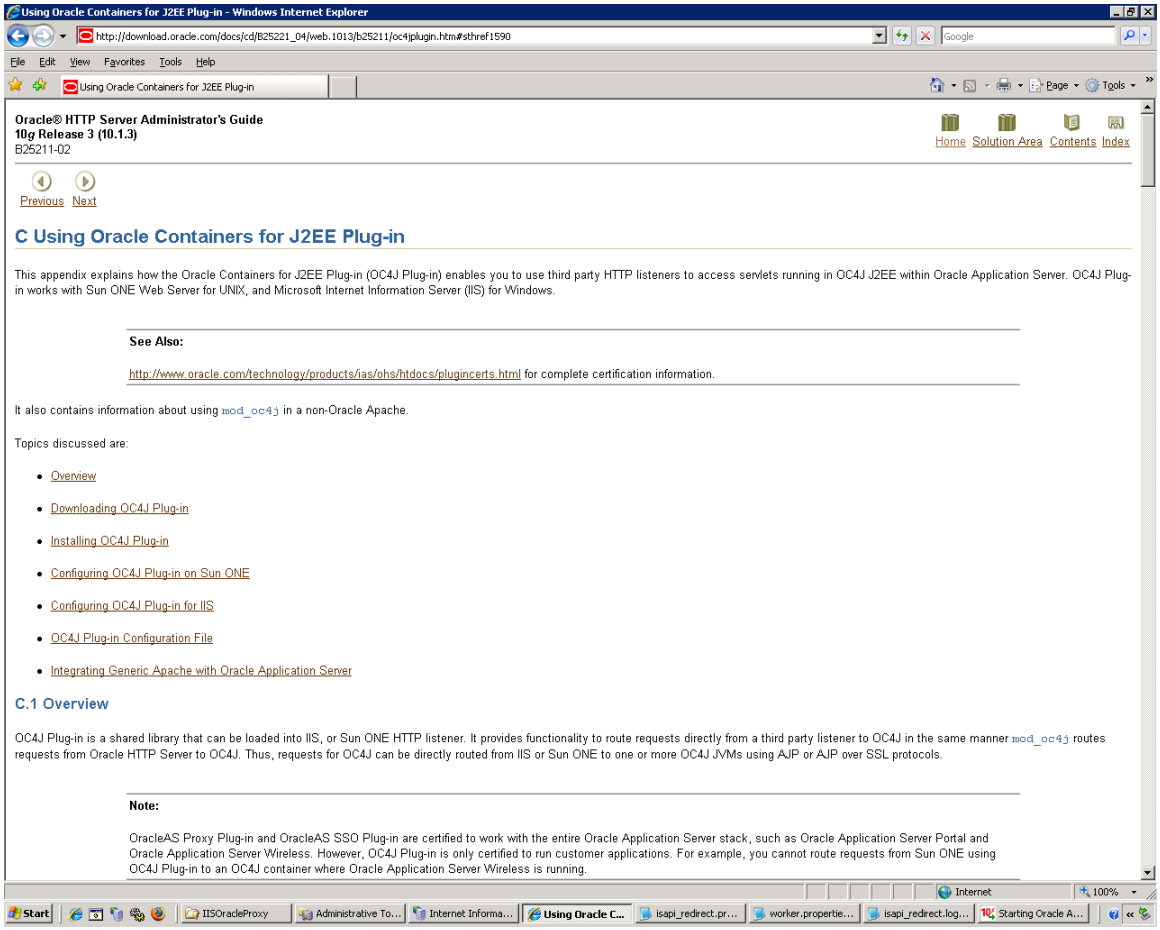
Case Study 2 – Oracle Application Server (OC4J) 10.1.3

Conceptually, OC4J is similar to Tomcat in the following diagram. It also offers an IIS Plug-in and internally uses AJP 1.3 protocol. However, it takes more time to configure due to its complexity.



(http://download.oracle.com/docs/cd/B25221_04/web.1013/b25211/oc4jplugin.htm#sthref1590)

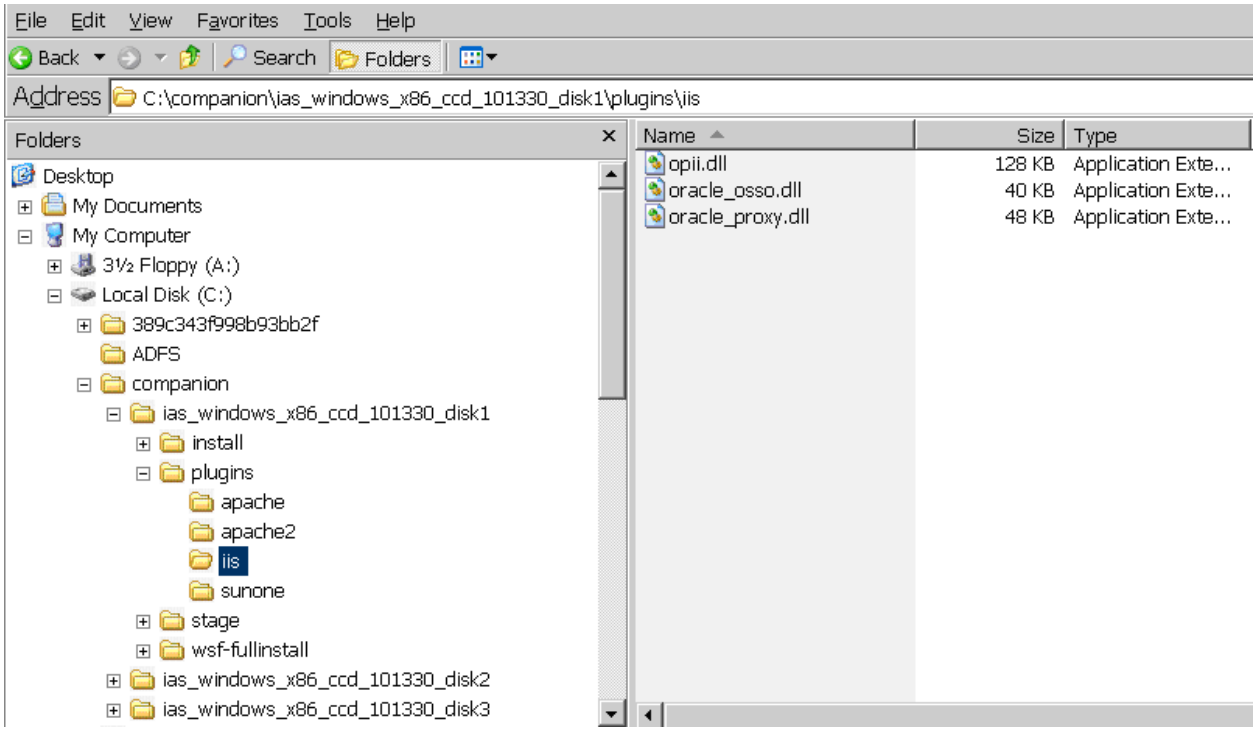
The documentation is Oracle HTTP Server Administrator's guide 10g release 3.



We need to download the needed software component before we begin.

- IIS Plugin for OC4J (**opii.dll**)

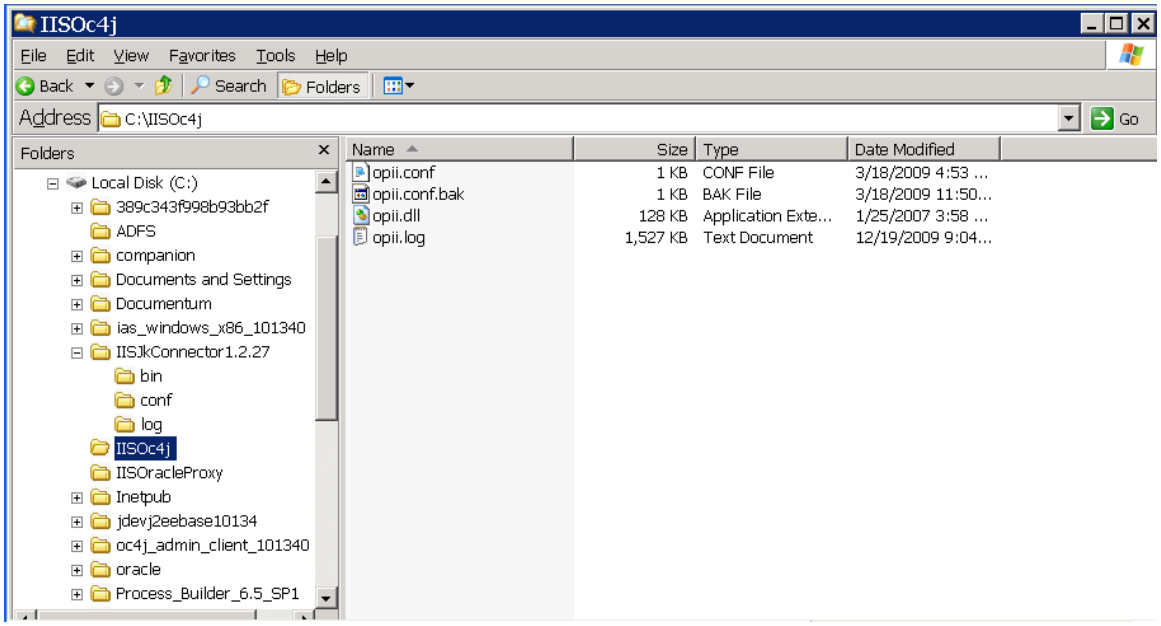
The source of this plug-in is in a companion CD for 10g, called **ias_windows_x86_ccd_101330_disk1,** that you can download from Oracle Technology Network (OTN) web site. After you unzip it, you should see the following structure (The "IIS" folder contains the Plugin):



Steps:

1. Setup IIS Windows Integrated Authentication

This step is identical to the aforementioned Tomcat case. I will skip it here. But to differentiate the two cases, we named this web application "**webtop2**", and created a folder, **C:\IISOc4j**



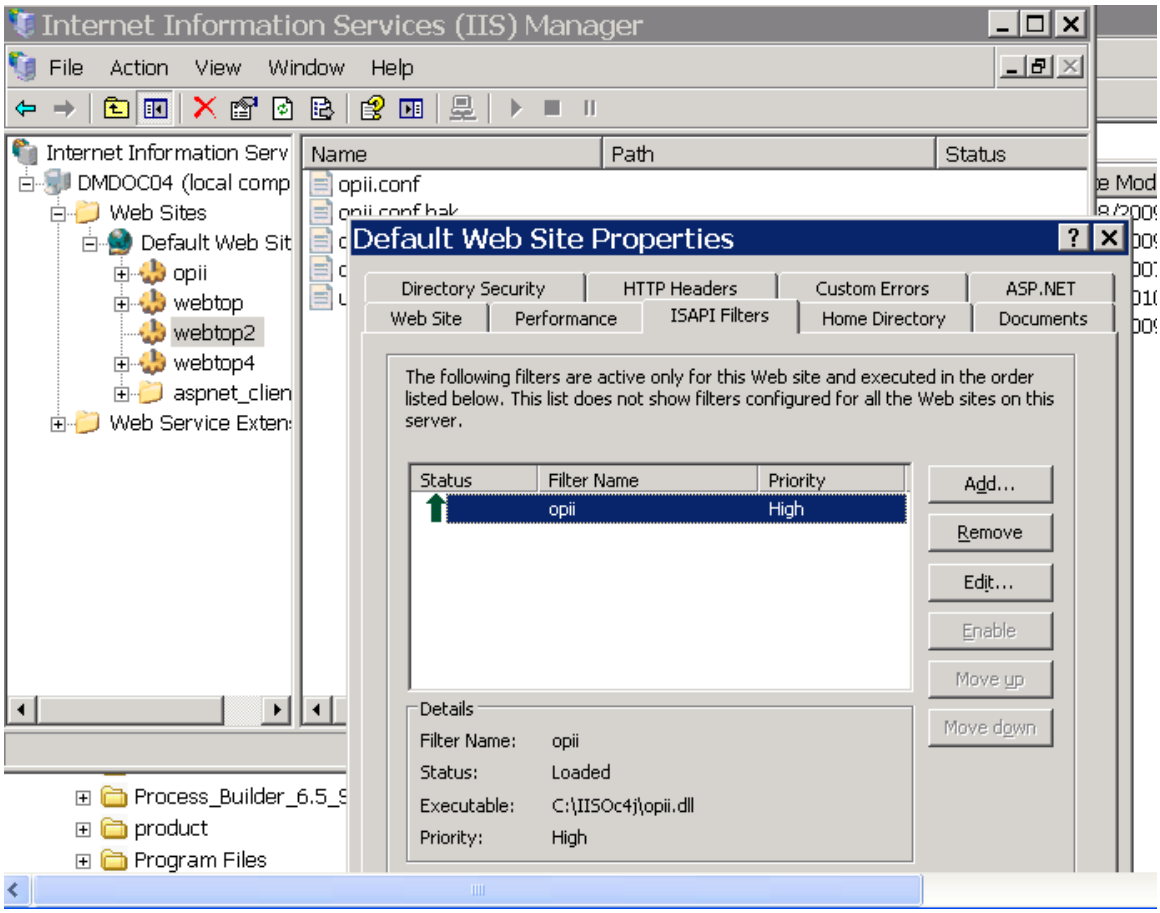
2. Test user.asp to make sure ASP remote user is valid

This step is identical to the Tomcat case. Please refer to it.

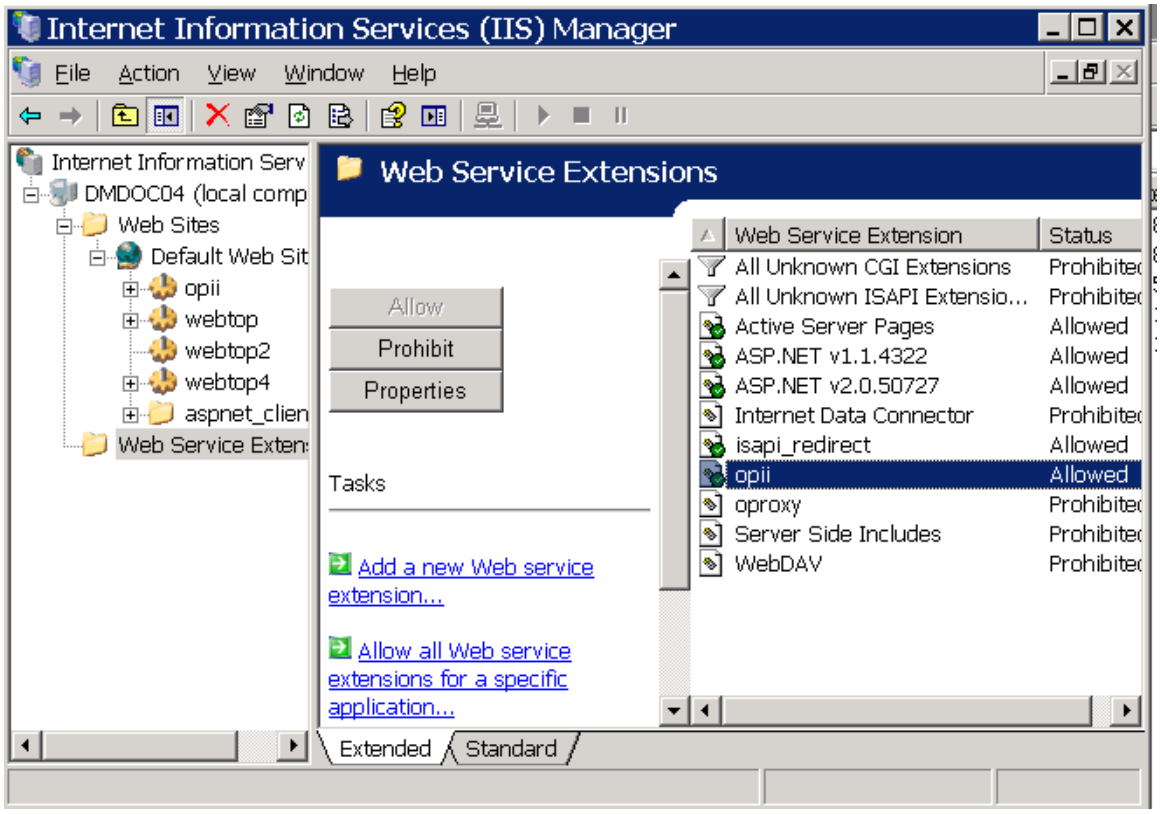
3. Setup IIS Plug-in for OC4J J2EE applications

To set up the IIS Plug-in, first, we need to setup an ISAPI Filter so that the Plugin DLL will be loaded when the IIS service starts up and intercepts the HTTP request.

We can go to the properties of “webtop2” and select tab “ISAPI Filter”, then add an “opii” ISAPI Filter and point it to **C:\IISOc4j\opii.dll**

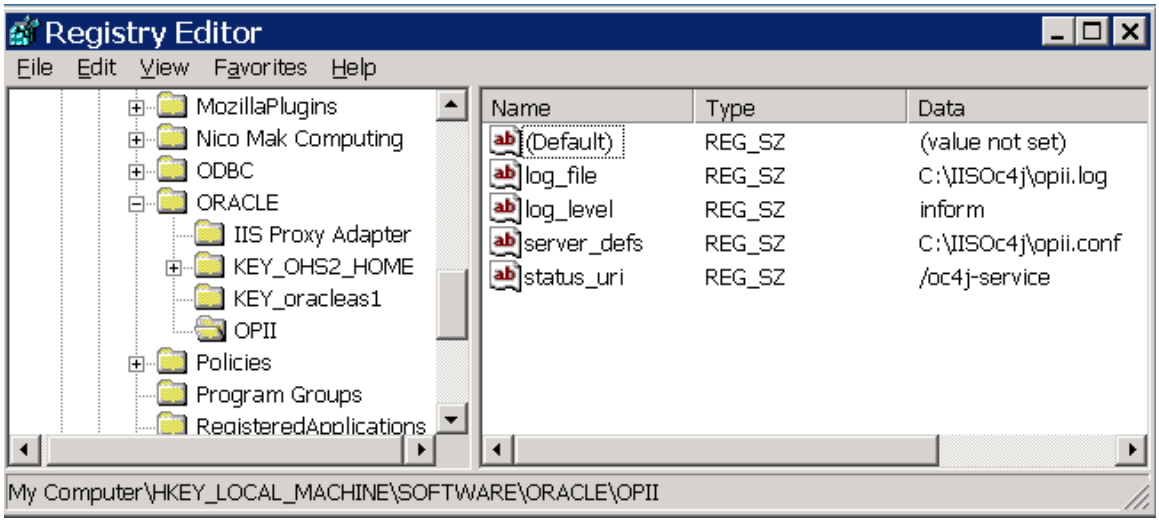


Also, make sure the opii filter DLL (opii.dll) is allowed in “Web Service Extension”. Oracle also requires a web virtual directory “**opii**” to be created as mapped to the folder where the “**opii.dll”** is located.



Next, we start configuring the Plug-in itself. Unlike the Tomcat case, the first configuration goes to the Windows Registry:

HKEY_LOCAL_MACHINE\Software\Oracle\OPII to create keys to point to the local file locations:



The required keys and values are:

Log_file = C:\IISOc4j\opii.log

Log_level=inform

Server_defs=C:\IISOc4j\opii.conf

Status_uri=/oc4j-service

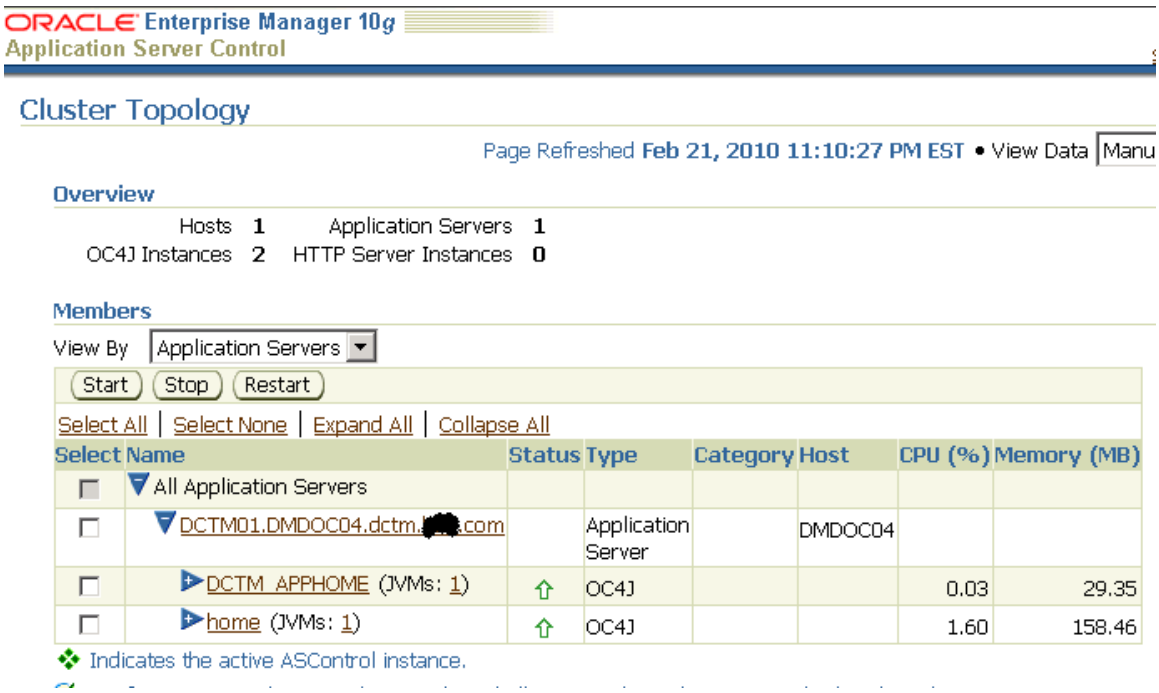
As can be seen from above, opii.conf is the main configuration file.

The file **opii.conf** contains the mapping and redirection information: (each Oc4jMount is a single line configuration string)

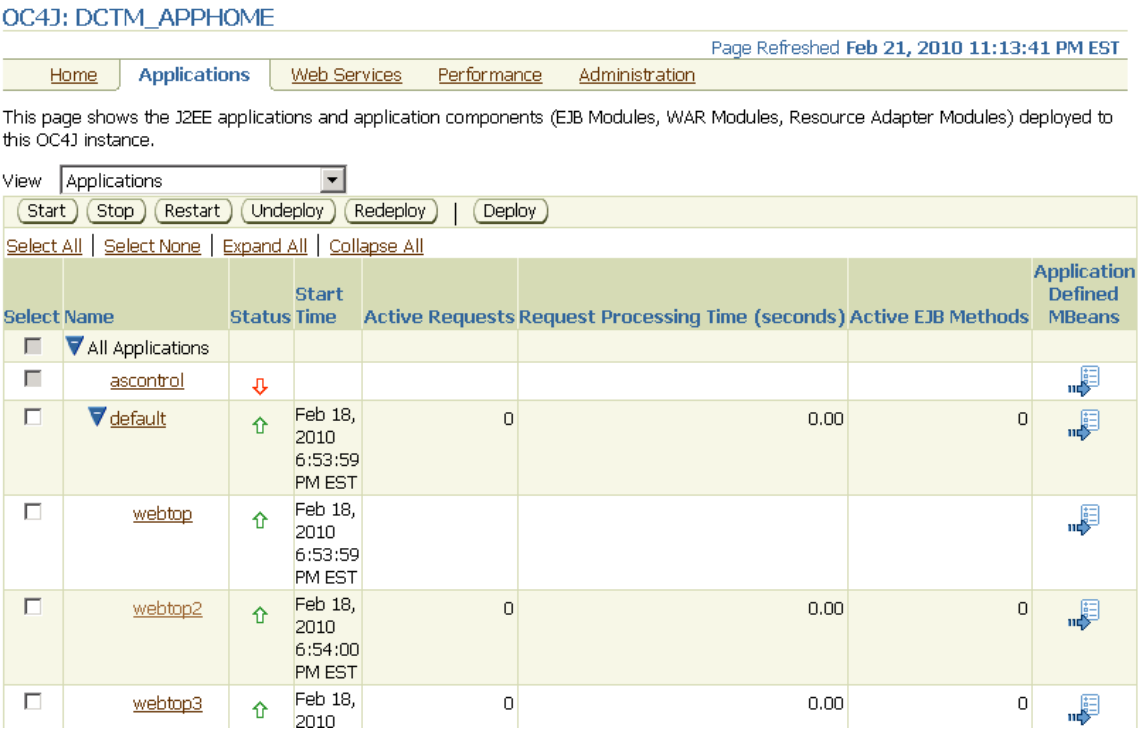
Oc4jMount /webtop2 instance://**DCTM01.DMDOC04.dctm.acme.com**:DCTM_APPHOME

Oc4jMount /webtop2/* instance://DCTM01.DMDOC04.dctm. acme.com:DCTM_APPHOME

"**DCTM01.DMDOC04.dctm.acme.com"** Is the OC4J application server instance name and "**DCTM_APPHOME"** should be the OC4J instance name that our webtop will be deployed to.



OC4J Instance, DCTM_APPHOME



4. Setup OC4J to verify AJP13 and ports

It is very easy to set up Tomcat but it's difficult to use OC4J for AJP13 because you don't see port settings from opii.conf. Obviously, Oracle's "Instance" protocol uses some internal port references for AJP13. Eventually, I managed to update **opmn.xml** located at:

C:\oracle\product\10.1.3.1\OracleAS_1\opmn\conf

The default web site protocol changed to "ajp" instead of http.

<port id="default-web-site" range="7777" protocol="**ajp**"/>

Then, when I checked the Oracle Enterprise Manger 10g, I noticed the protocol change. I have taken out the HTTP and replaced it with AJP since we do not need built-in HTTP.

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology >

Runtime Ports

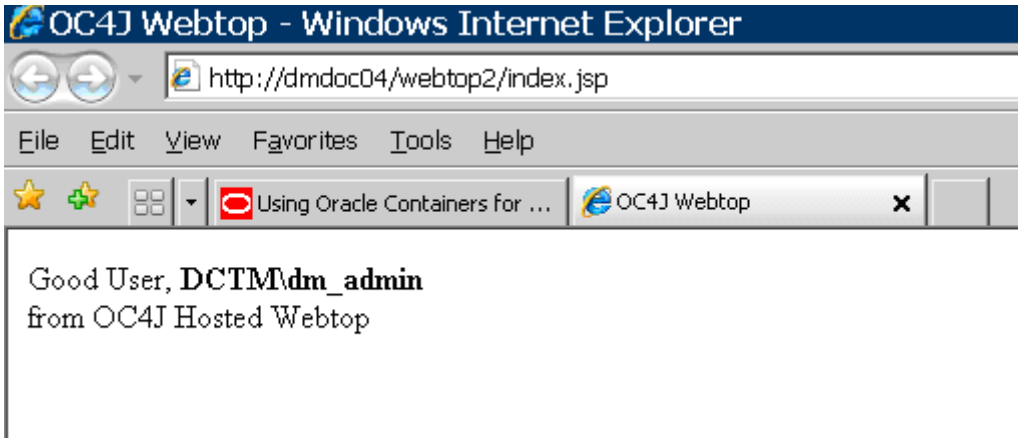
Page Refreshed Feb 21, 2

Expand All | Collapse All

Name	Host	Port In Use	Port Range	Configure Port
All Application Servers				
DCTM01.DMDOC04.dctm.[redacted].com	DMDOC04			
OPMN				
Local		6100	6100	
Remote		6200	6200	
Request		6003	6003	
OC4J : DCTM_APPHOME				
JMS		12602	12601-12700	
AJP		7777	7777	
RMIS		12702	12701-12800	
RMI		12402	12401-12500	
OC4J : home				
JMS		12601	12601-12700	
HTTP		8888		
RMIS		12701	12701-12800	
RMI		12401	12401-12500	

5. Test index.jsp to make sure JSP remote user is valid

This step is critical yet identical to Tomcat case except I named the test file "**index.jsp**".



This demonstrates the Remote User redirection works with IIS, OPII and OC4J.

6. Configure Customized Webtop login.jsp

This step is identical to Tomcat case.

Conclusion

This article discussed a Windows integrated SSO solution for Documentum Webtop application. The general idea was endorsed by the consultants from Microsoft, Oracle, and EMC Documentum. One of the implementations based on this is in production. You can get immediate benefits by following these implementations. The implementations mentioned in this article, however, are just the beginning. Hopefully, it would help and inspire you to expand the use of Documentum to other external systems, Web Servers and Application servers. For example, SharePoint apparently supports Kerberos and Windows Integrated authentication. Apache also support Kerberos and KDC. On the Java side, we have JBoss, Glassfish, Websphere, etc. The list goes on. Happy Documentuming!

References

1. Nancy Chamberlin, Department of Justice, “a Brief Overview of Single Sign-On Technology”, A view to the future, Government Information Technology Issues 2000
2. Wikipedia, “Integrated Windows Authentication”
3. http://support.microsoft.com/kb/822248
4. http://technet.microsoft.com/en-us/library/cc780455(WS.10).aspx
5. IIS Authentication, http://msdn.microsoft.com/en-us/library/aa292114(VS.71).aspx
6. Microsoft. "Integrated Windows Authentication (IIS 6.0)". IIS 6.0 Documentation
7. EMC Documentum Content Server Fundamentals version 6.5, P/N 300-007-197-A01
8. EMC Documentum Content Server version 6.5, Administration Guide, P/N 300-007-198-A02
9. EMC Documentum Foundation Classes, version 6.5, Development Guide, P/N 300-007-210-A01
10. http://download.oracle.com/docs/cd/B25221_04/web.1013/b25211/oc4jplugin.htm#sthref1590