Native LDAP/AD authentication with NodeJS server

Admittedly not the most appealing title for an article, but it's somewhat difficult to find an alluring tag line when it comes to authentication! Still, did you know that the functionality to authenticate users through an LDAP/AD server before they can submit any job to Workflow can be enabled natively with just a few clicks? If you didn't, then please read on!

What's LDAP/AD?

For the sake of this article, LDAP (*Lightweight Directory Access Protocol*) and AD (*Active Directory*) are the same thing. The former is a standard protocol for authentication and policies, while the latter is Microsoft's take on LDAP. There are some differences, of course, but none pertain to this article. Therefore, to make for easier reading, we will simply refer to both as *LDAP*.

Where is LDAP used, and what for?

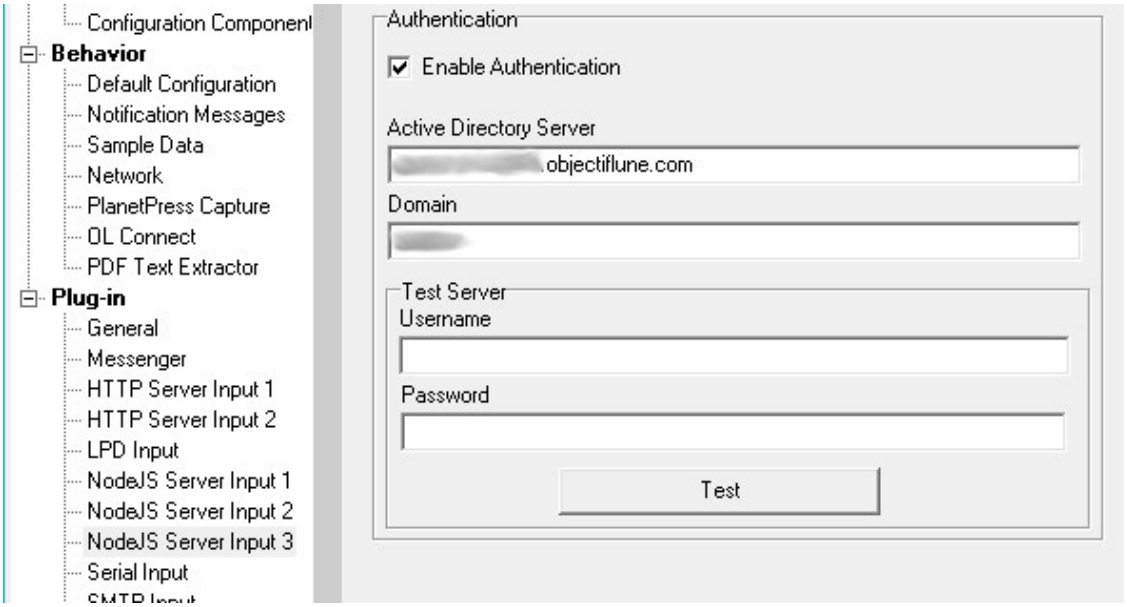
LDAP is used... um... everywhere? Seriously, most organisations have one or more LDAP servers that handle user names and accounts, along with roles and directory access policies. When you log into your organisation's intranet or into your corporate email account, the username and password you enter are most likely being handled and validated through an LDAP server. This authentication also includes a list of things you are allowed - or not allowed - to do on that intranet or that corporate mail account.

OK, so how is this important or useful?

Well in many implementation of OL Connect, the organization wants to make sure that some operations can only be triggered by trusted people... and the best way to make sure of that is by requiring users provide their login credentials. For instance, viewing a web-based dashboard that contains some financial or HR data is definitely not something you'd want to share with people outside the organization, and you might even want to restrict it to a certain group of people within the organization. That's where LDAP authentication can help you implement a trusted connection with the user accessing the service.

How can LDAP be used with Connect?

LDAP authentication is integrated with the NodeJS server that ships with OL Connect Workflow. It is enabled through the **Workflow Preferences | NodeJS Server Input 3** section. All you need to do is tick the box and specify your LDAP server name/address and your domain name:



Only those three parameters are required for LDAP authentication to be applied to all NodeJS input requests (exceptions are discussed later in this article). The two additional fields are only used to help you validate that the domain and LDAP server names are valid. Use known credentials and press the *Test* button : Workflow will respond with a success message if it is able to authenticate those credentials. But at run time, those credentials are not used.

If you don't know what your LDAP server and domain names are, you can *usually* obtain them by opening a CMD window and typing the following:

For the domain name:

```
echo %USERDOMAIN%
```

For the LDAP Server name:

```
echo %logonserver%
```

This last command may return a value prefixed with backslashes (e.g. `\\MyDomain`), which must be removed before setting the *Active Directory Server* field displayed above. If the values obtained from the above commands don't work for you, give your IT department a call. They will definitely know!

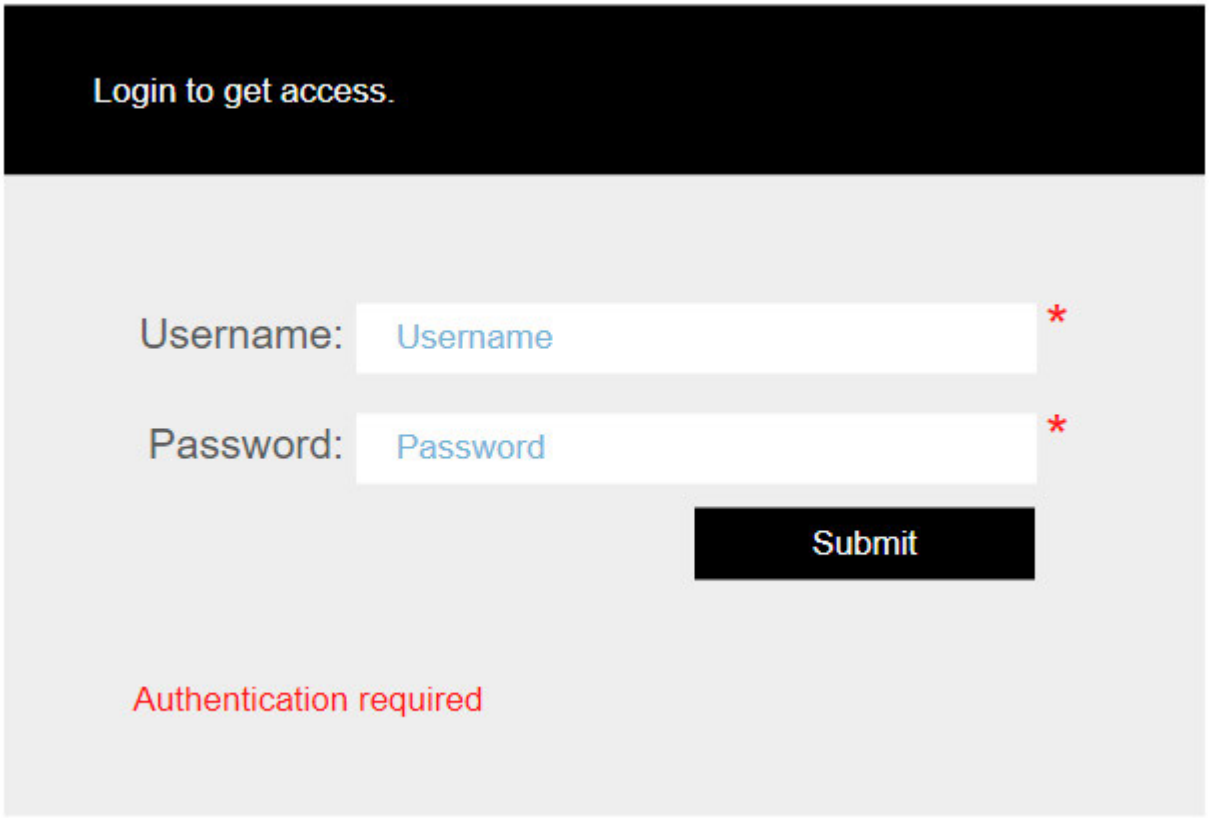
What happens next?

Once you have saved your preferences and sent them to the Workflow Service, any process that starts with a NodeJS input task will require the user to authenticate before it can run. Let's imagine you have a simple web

process named *MyTestProcess* that returns a basic web page showing "Hello World!". If you open a browser window on the same PC where Workflow is installed and type the following URL:

```
http://127.0.0.1:9090/MyTestProcess
```

you will not immediately get the "Hello World!" message. Instead, you will get this login dialog window:



Once you have filled out the fields with valid credentials, you are taken to the "Hello World!" page. If you open a second browser window and type in the same URL, you will not be asked to log in once again because your previous authentication is valid for the duration of the session. If you close all browser windows and then re-open a new one and navigate to the same URL, you will be asked to log in once more.

Note: it is important to understand that until the user has been successfully authenticated, the Workflow process is never triggered. So even if someone were to repeatedly attempt to brute-force their way into executing a Workflow process, those attempts would never reach Workflow itself as they would be stopped at the NodeJS level. This makes the entire system much more secure. Note also that after a certain number of failed attempts, the NodeJS server will lock out the user for a certain length of time, to discourage denial of service attacks.

Can the Authentication last across sessions?
There is no option in the Workflow Preferences that allows you to set a different behavior for the duration of the authentication. However, you can manually edit the file named:

```
C:\Program Files (x86)\Objectif Lune\ppnode\src\constants\default.js
```

Look for the line `exports.DEFAULT_SESSION_TTL = 0;` and change the value to the number of milliseconds that you want the authentication to last. **Note**: be ***very*** careful when you edit that file, messing with any other value may result in unpredictable behavior!

Can I change the look and feel of the login window?
Well... yes... but only attempt to do so if you are familiar with the **EJS** templating syntax. There are plenty of tutorials available online about this syntax, and it is pretty simple to grasp (especially for dialog windows as simple as a login screen), but the same warning as above applies: be ***very*** careful when editing those files, messing with them may result in unpredictable behavior!

The two files files are located here:

```
C:\Program Files (x86)\Objectif Lune\ppnode\src\html
```

How can Workflow use credential information?
It must be stressed that *at no time* do NodeJS and Workflow know what your password is. That's the beauty of the LDAP connexion: values are passed directly to the LDAP server, who responds with authentication cookies that are set to last for the duration specified in the **default.js** file. Once the user has been authenticated, the NodeJS server stores those cookies inside the request file that it generates and hands over to Workflow. It also adds to the request file the user name of the user who successfully logged on (under the aptly named `<username>` element).

Now if you want to use that information, you'll need a script like the one below, which uses the user name value to query the LDAP server for additional information about the user:

```
// Set up the various objects required for connecting to the LDAP server
var objConnection = new ActiveXObject("ADODB.Connection");
objConnection.Open("Provider=ADsDSOObject");
var objCommand = new ActiveXObject("ADODB.Command");
objCommand.ActiveConnection = objConnection;
objCommand.Properties("Page Size") = 1000;

var userName = Watch.Expandstring("");

// Query the LDAP server for user info
objCommand.CommandText ="SELECT name, department, sAMAccountName From 'LDAP://olca1-
dc4.objmtl.objectiflune.com/DC=objmtl,DC=objectiflune,DC=com' WHERE objectClass='User'
and sAMAccountName='"+userName+"'";
var objRS = objCommand.Execute;

if(!objRS.EOF){
objRS.MoveFirst;
var userObject = {};
userObject.name = objRS.Fields("name").Value;
userObject.ID = objRS.Fields("sAMAccountName").Value
userObject.department = objRS.Fields("department").Value
Watch.Log(JSON.stringify(userObject),2);
}

objRS.Close();
objConnection.Close();
```

This script retrieves the user's full name and the Department to which she/he belongs. Presumably, the rest of the Workflow process could use this additional piece of information to determine whether or not members of one or more departments are allowed to access the process.

EDIT (2020-03-31): using a fully qualified name for the LDAP Server connection string results in a much faster call. The original script used simply "`LDAP://olca1-dc4`" but it turns out that the fully qualified path "`LDAP://olca1-dc4.objmtl.objectiflune.com/DC=objmtl,DC=objectiflune,DC=com`" resolves much faster.

What if only a few processes require authentication?
That's the easy part: you set the NodeJS server's preferences to use authentication just like we did above, but in each NodeJS Input task that doesn't require it, you simply tick the option "Ignore global authentication settings" and *voilà* : no more login required for that specific process.

What have we learned?
When you have an Active Directory or LDAP server in place for authentication purposes, there is no need for you to build your own authentication system or create your own database of users. You can use the native functionality of OL Connect Workflow's NodeJS server to build business rules based on the corporate policies provided by the LDAP server.

This authentication process can be used as long as the entry point into Workflow is the NodeJS Server Input. That means you could also build an interactive OL Connect Remote Print (aka OL Connect Send) that authenticates users before accepting their jobs.