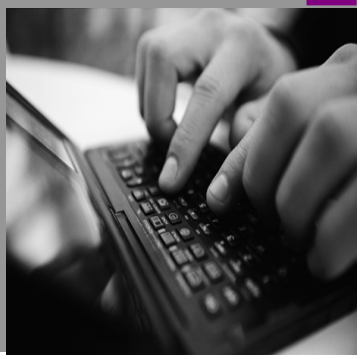
How-to Guide
SAP NetWeaver '04



How To... Authentication Guideline

Version 1.00 – May 2004

Applicable Releases:
SAP NetWeaver '04
(User Management Engine)

1 Introduction

SAP Enterprise Portal 6.0 provides a flexible mechanism for plugging in custom authentication mechanisms and fine-tuned authentication configuration for the content running within the portal.

This guide explains the basic concepts where the customer can plug in his own code, how he can make use of the flexible architecture and how all the different pieces come together.

This guide does not explain what JAAS[1] is and how it works. If you are not familiar with JAAS authentication, please read [1] first. Furthermore, the reader is supposed to be an experienced JAVA programmer. Useless to say that it is very helpful to read SAP documentation and/or whitepapers to know about the basic concepts of the SAP Enterprise Portal, iViews, authschemes, etc.
The structure of this guide is as follows: In section 2 we cover the simplest way to customize the logon: a LoginModule. Only by writing a LoginModule and some small configuration changes you can already alter the way authentication is performed in the portal. The portal will then use the standard authentication screen when prompting the user. In 2.5 we show how to replace the standard portal login screen by a screen developed on your own. Section 3 gives you insight in the more advanced features of the architecture: how to evaluate http information and how to deal with cookies.

2 Custom password based authentication

2.1 Use case

Password based authentication is the most common authentication method in software systems with a user administration and authentication requirement. SAP Enterprise Portal already provides numerous password based authentication features: Password validation against all supported LDAP directories, against the UME 4.0 proprietary database repository and against an ABAP engine. However, this may not be enough in your scenario.

There are various scenarios imaginable where a custom authentication is necessary. To give you only two examples: a) You might have a proprietary user repository (e.g. LDAP compliant) where user records can be synchronized with the portal user repository but passwords cannot. This can easily happen if the one-way hash algorithm used to store passwords is not identical in both user repositories (then you cannot simply copy the stored hash values from one repository into the other nor can you restore the password from the hash value). The Enterprise Portal user administration can then use the synchronized user record in the portal user repository but the authentication must be performed using the original repository. This is where the LoginModule comes into the picture. b) You are using a hardware device that generates one time pins for authentication. Generally, these hardware devices are synchronized with some central server which verifies the pins. In this case the LoginModule would establish a connection with the central server and verify the pin.

However, we're not going to tackle a complex task like this. The purpose of this document is to demonstrate the basic principle and what needs to be done for integration into SAP Enterprise Portal. Therefore, we don't spend a lot of time on the user credential verification and focus on the other issues.

[1] See [1]

2.2 From the authenticating programming code to the login module

As mentioned above our focus is not the mechanism for password verification. Therefore we've decided to use a simple mechanism for demonstration purposes: We'll accept the password if it is the cyclic right-shift by one character of the user name. We'll ignore uppercase/lowercase when doing the comparison for reasons of simplicity.

2.2.1 The programming code that does the authentication

So there is a function

```
checkPwd (String user, char [] pwd)
```

in the LoginModule. Please note that the password comes as a character array in compliance with the way passwords are handled in the JAAS standard.

Here's the complete code of the method:

```
    /**
     *  This function verified if the password is a cyclic
     *  right shift by one character of the lowercase user
     *  name
     *  @param mycallbacks
     *  @return
     */
    protected boolean checkPasswd (String name, char [] pwd)
    {
        if (name==null || pwd==null || name.length()==0 ||
pwd.length==0) {
            return false;
        }
        name = name.toLowerCase();

        // cut off the first character
        String strPwd = new String (pwd, 1, pwd.length-1);

        // compare if the first character of the password
        // is the last of the user name
        if (pwd[0]!=name.charAt(name.length()-1))
            return false;

        return name.substring (0,name.length()-1).equals(strPwd);
    }
```

2.2.2 The Login module

Here's the rest of our login module:

Attributes:

```
Subject         _subject    = null;
CallbackHandler _ch         = null;
```

Initialization:

```
    public void initialize(
```

```
            Subject subj,
            CallbackHandler ch,

            Map sharedState,
            Map options)
        {
            _subject = subj;
            _ch      = ch;
        }
```

Login:

```
        public boolean login()
            throws LoginException
        {
            Exception        exception_on_the_way    = null;
            PasswordCallback pc = new PasswordCallback ("Password:",
    false);
            NameCallback     nc = new NameCallback ("User:");
            Callback []      mycallbacks = new Callback [] { nc, pc };

            try {
                _ch.handle (mycallbacks);
            }
            catch (IOException e) {
                exception_on_the_way = e;
            }
            catch (UnsupportedCallbackException e) {
                exception_on_the_way = e;
            }

            String name = nc.getName();
            char [] pwd = pc.getPassword();

            if (name.length()==0)
                throw new LoginException (MISSING_UID);

            if (pwd.length==0)
                throw new LoginException (MISSING_PASSWORD);

            if (exception_on_the_way!=null) {
                // In the real world we'd write an entry
                // into the trace here
                exception_on_the_way.printStackTrace ();
               throw new LoginException ("Couldn't handle callbacks");
            }

            if (!checkPasswd (name, pwd)) {
                throw new LoginException (USER_AUTH_FAILED);
            }
            else {
                _bSucceeded = true;
                _auth_user  = name;
            }

            return true;
    }
```

Commit:

```
public boolean commit ()
{
    if (_bSucceeded == false) {
        return false;
    }
    else {
    // add a Principal (authenticated identity) to the Subject
        final String final_name = _auth_user;
        _subject.getPrincipals().add (new Principal () {
             public String getName ()
             {
                 return final_name;
             }
        });
        return true;
    }
}
```

Needless to say that this is the most simplified implementation. However it demontrates how you get user id and password and how you validate it and how you provide the identity of the authenticated user to the outside world.

2.2.3 What you need to pay attention to

There are two important items you need to know about.

1. **User names.** The user name you create the `Principal` object with needs to be the logon uid of a user. More precisely, SAP Enterprise Portal will try internally to instantiate a `com.sap.security.api.IUser` object with the method IUserFactory.getUserByLogonUid(String logonuid)[2].
2. **The sharedState parameter in the method `LoginModule.initialize()`.** For memory reasons the portal is not able to keep login contexts. Therefore, you won't be able to save settings throughout the entire logon/logoff cycle within the sharedState map. More precisely, a new sharedState map is instantiated before `login()` or `logoff()` are called. You'll need to use a different mechanism.
3. **Error codes.** If you throw a `LoginException` from within the method `login()`, please note that the standard login page displays standard error messages for the standard error cases like "user of password incorrect". The mechanism to tell the UI to display a certain message is to throw a `javax.security.auth.login.LoginException` with a specified error constant that is recognized by the UI. These (String) constants are defined in the core class `com.sap.security.core.logon.imp.SecurityPolicy`. In addition to the fact that this class is a core class and therefore not part of the published UME API the DemoLoginModule code would need a reference to the UME libraries which results in a cyclic class loader dependancy. This is to be avoided. Therefore we redefine the error codes. This is not a nice solution but at least it works. For a list of the most important error codes see 2.2.4. In case you do not use the UM Uis, you are free to define your own error codes (see section 2.5).

2.2.4 Error codes

```
public final static String MISSING_UID      = "MISSING_UID";
public final static String MISSING_PASSWORD = "MISSING_PASSWORD";
public final static String USER_AUTH_FAILED = "USER_AUTH_FAILED";
```

[2] see also [2]

```
public final static String USERID_NOT_FOUND = "USERID_NOT_FOUND";
public final static String ACCOUNT_LOCKED_ADMIN =
"ACCOUNT_LOCKED_ADMIN";
```

```
public final static String ACCOUNT_LOCKED_LOGON =
"ACCOUNT_LOCKED_LOGON";
```

2.3 Configure the portal

There are some changes necessary to make your changes available in the portal. You need to make the code visible in terms of class reference visibility and you need to configure the portal to make your LoginModule the one that is called within an authscheme.

2.3.1 Making the jar available

Build a Java archive (jar) file containing your loginModule and all other classes you need. Place ths jar file (in our example demolm.jar) into the folder *<j2ee home>/cluster/server/additional-lib*.

2.3.2 Changes in library.txt

Your classes need to be visible by the UME classes and vice versa. Due to the complex classloader infrastructure in the J2EE engine the easiest and less risky solution is to add your jar file to the definition of the UME libraries in *<j2ee home>/cluster/server/managers/library.txt*. Add the name of your jar file to the line

```
library com.sap.security.ume
com/sap/security/api/com.sap.security.api.jar;com/sap/security/api/com.sa
p.security.api.perm.jar;com/sap/ip/basecomps/BaseComps.jar;com/sap/securi
ty/api/com.sap.security.core.jar;com/sap/security/api/com.sap.security.co
re.tpd.jar
```

this would look like this:

```
library com.sap.security.ume
com/sap/security/api/com.sap.security.api.jar;com/sap/security/api/com.sa
p.security.api.perm.jar;com/sap/ip/basecomps/BaseComps.jar;com/sap/securi
ty/api/com.sap.security.core.jar;com/sap/security/api/com.sap.security.co
re.tpd.jar;com/sap/security/testlm.jar;testlm.jar
```

if the name of your jar file is testlm.jar.

2.3.3 Changes in authschemes.xml

In order to plug in the login module into the login cycle of Enterprise Portal you need to create a new authscheme. In authschemes.xml, add a new authscheme section

```
<authscheme name="myNewLogon">
  <loginmodule>
    <loginModuleName>
      com.sap.security.demo.DemoLoginModule
    </loginModuleName>
    <controlFlag>REQUISITE</controlflag>
    <options></options>
  </loginmodule>
  <priority>20</priority>
  <frontendtype>2</frontendtype>
  <frontendtarget>com.sap.portal.runtime.logon.certlogon</frontendtarget>
</authscheme>
```

This authscheme is identical to the standard authscheme “uidpwlogon” except for the name of the logon module.

Now you have two options. The first option is to assign the authscheme “myNewLogon” only to some iViews in your portal. The second option is to make “myNewLogon” the standard authscheme. To do this, you simply change the authscheme reference “default” from “uidpwlogon” to “myNewLogon”:

```
<authscheme-refs>
  <authscheme-ref name="default">
    <authscheme>myNewLogon</authscheme>
  </authscheme-ref>
</authscheme-refs>
```

You can find the entire authschemes.xml in the list of attached files. (In order to save changes to authschemes.xml when you upgrade Enterprise Portal, please see SAP note 686538).

2.4 The entire example at work

These are all necessary changes. After all changes have been made restart your server and log on to the regular portal logon.

2.5 Your own login screen

In order to write your own login screen you need to write a Java iView with certain properties. We are not going through the entire process of writing an iView here. Instead, we just highlight what you the above mention “properties” are.
We suppose that you are able to write an iView with a jsp page.

2.5.1 Necessary html-form parameters

If you intend to customize the login screen you need to know which parameters to submit. In your HTML form the name of the user parameter must be `j_user` the name of the password parameter `j_password`. If you want to process a logon for another authscheme than the default authscheme you also need to provide the name of the authscheme. The parameter name is `j_authscheme`.
In addition to these parameters you need to add a parameter `login_submit` with value true to the request to signal the portal that a logon takes place.

2.5.2 The jsp page

For an example of such a page you should have a look at the standard logon page (see the below simplified code):

```
<FORM name="logonForm" method="post" action="
/irj/servlet/prt/portal/prtroot/com.sap.portal.navigation.portallauncher.default">
  <input name="login_submit" type="hidden" value="on">
  <input type="hidden" name="login_do_redirect" value="1" />
  <input name="j_authscheme" type="hidden" value="default">
  <table border="0" align="left" valign="top">
    <tr>
      <td >
          User Id
      </td>
      <td width="183" height="20">
        <input name="j_user" type="text" value="">
      </td>
    </tr>
    <tr>
      <td width="161" height="20">
          Password
        </label>
      </td>
      <td width="183" height="20">
```

```
        <input name="j_password" type="password">
      </td>
    </tr>
    <tr>
      <td colspan="2" height="20">  </td>
    </tr>
    <tr>
      <td colspan="2">
        <input value="Log on">
      </td>
    </tr>
  </table>
</form>
```

Comments:

1. By changing the j_authscheme parameter to the name of another authscheme you can determine which authscheme is going to be processed.
2. If

3 A more advanced example

The portal allows access to the entire http request during authentication to all custom login modules. You can read and set cookies or http header values, for instance.

3.1 Use case

Section 2 only handles the most basic case. In many cases there are more sophisticated requirements for an authentication decision, for instance reject certain ip address ranges. Accessing http header variables requires a little more work, we'll cover that in this section.

3.2 The Login module code

The login module differs from the one in 2.2 in that an additional check is performed. This additional check is whether the ip address of the client is in a given ip range. "Additional" means that the password still needs to be correct (in our simple example: a cyclic shift of the user id), so we can derive a new LoginModule class from the one in 2.2 and overwrite the method `login ()`.
Another difference is that we need to read a parameter from the configuration of the LoginModule. Therefore, we store a reference to the options parameter in the initialization method `initialize()`.
Here's the code:

3.2.1 Attributes

```
Map _options;
```

3.2.2 The method login()

```
Exception on_the_way    =   null;
String    client_ip_str =   null;
String
ip_range_str=(String)_options.get("com.sap.security.demo.ip_mask");
String ip_match
=(String)_options.get("com.sap.security.demo.ip_match");

// We check anyway if the old login
// succeeds or not
boolean rc = super.login ();

// In addition we perform the check whether
// The ip address is within a given range
```

```
// Get ip-range from options
byte [] iprange_as_byte_array = getIPAsByteArray (ip_mask);

// Get ip-address of client

byte [] client_ip = null;
// Get ip-match
int [] ipmatch = getIPAsIntArray (ip_match);

try {
    client_ip_str =   getClientIP ();
}
catch (UnsupportedCallbackException e) {
    on_the_way = e;
}
catch (IOException e) {
    on_the_way = e;
}

client_ip = getIPAsByteArray (client_ip_str);

if (on_the_way!=null) {
    on_the_way.printStackTrace ();
    throw new LoginException ("Exception occured");
}

if (!ip_address_in_range (client_ip, iprange_as_byte_array,
ipmatch))
    throw new LoginException ("IP-address " + client_ip_str + " is
not in
     range " + ip_range_str);

return rc;
```

3.2.3 Utility methods

The method `ip_address_in_range()` checks whether the client ip is in a given range. Therefore, it first flips all bits in the array `iprange_as_byte_array` and performs then a logical AND with the ip address. The result is that whenever there is a bit set outside of the allowed range, this bit will be preserved. All bits within the allowed range are cleared. Therefore the ip address is ok if and only if the result is 0.
Please note the method `getClientIP()`. In order to get access to all data in the http request, call `handle()` with an instance of `WebCallback`.

```
private boolean ip_address_ok (byte [] client_ip,
                               byte [] iprange_as_byte_array,
                               byte [] ipmatch)
{

    for (int ii=0; ii<4; ii++) {
        if (ipmatch[ii]!=(client_ip[ii] &
iprange_as_byte_array[ii]))
            return false;
    }

    return true;
}
```

```
   private String getClientIP ()
       throws UnsupportedCallbackException, IOException
   {
       WebCallback wcb = new WebCallback ();

       _ch.handle (new Callback [] { wcb });


       HttpServletRequest req = wcb.getRequest();

       return req.getRemoteAddr ();
   }
```

3.3 Configuration

Now we have a second login module that we can add to the xml definition of authschemes. Here's the corresponding section that's added to the xml file:

```
<authscheme name="myNewLogon2">
  <loginmodule>
    <loginModuleName>
      com.sap.security.demo.DemoLoginModule2
    </loginModuleName>
    <controlFlag>REQUISITE</controlflag>
    <options></options>
  </loginmodule>
  <priority>25</priority>
  <frontendtype>2</frontendtype>
  <frontendtarget>com.sap.portal.runtime.logon.certlogon</frontendtarget>
</authscheme>
```

The only difference compared to the authscheme defined in 2.3.3is the name of the authscheme, the class name of the login module and the priority. Note that the we consider this authentication mechanism slightly stronger than "myNewLogon", so we give it the priority 25.

4 References

[1]JAAS documentation – http://java.sun.com/products/jaas/
[2] UME API – see SAP Netweaver Developer Studio