StackableAuthenticationMethods

Stackable Authentication Methods

This page proposes an extension to the core DSpace classes to implement *stackable* authentication methods. By stackable, I mean the authenticators are deployed in a "stack" and tried each in order, instead of configuring a single authentication method.

Rationale

The reasons for rearranging the code this way are:

- Separate authentication from the Web user interface so the same code can be used for e.g. non-interactive Web Services.
- Improved modularity: No authenticator needs to know about any other authenticator. Custom authentication methods can be "stacked" with the default DSpace username/password method.
- Cleaner support for "implicit" authentication where username is found in the environment of a Web request, e.g. in an X.509 client certificate.

Implementation

The

```
AuthenticationMethod
```

interface encapsulates everything that should be needed to add a new method. It should not be necessary to change the UI, although it may be desireable to add different instructional messages to the pages that prompt for a username and password, if they are used.

```
public interface AuthenticationMethod {
  // symbolic return values
  public final int SUCCESS = 1,
                   BAD_CREDENTIALS = 2,
                   NO_SUCH_USER = 3,
                   BAD_ARGS = 4;
  // allow users to create new ePerson; this might consult configuration.
  public boolean canSelfegister();
  // allow self-registering user to change their password.
  public boolean allowSetPassword();
  // gets credentials from X.509 Cert. or other implicit means.
  public boolean isImplicit();
  // return IDs of groups the user is in implicitly.
  public int  getSpecialGroups(Context context, HttpServletequest request);
  // attempts to authenticate a user and sets ePerson in the context.
  // any of username, password, or request may be null when not available.
  public int authenticate(Context context,
                          String username,
                          String password,
                          HttpServletequest request);
  // returns UL to which to redirect to obtain credentials (either password
  // prompt or e.g. HTTPS port for client cert.); null means no redirect.
  public UL redirect(Context context,
                     HttpServletequest request,
                     HttpServletesponse response);
}
```

Now the Web UI first calls each of the authentication methods in the "stack" in turn until one succeeds. The complete algorithm for an interactive UI is:

1. Try each method for which

   ```
   isImplicit()
   ```

   is true, accepting the first one that succeeds, and redirect to the page that needed authentication.

2. If all of those fail, step through all methods trying the

   ```
   redirect()
   ```

   method. Return a Web redirect to the UL of the first non-null return – which is presumably a form page to request authentication credentials(i.e. username/password).
3. That page will trigger another authentication attempt though the appropriate servlet, which is effectively part of the authentication method. It (hopefully) authenticates this time, and then redirects to the page that triggered the authentication request.

There should also be a final sanity check; if the

```
EPerson.getequireCertificate()
```

is true,
it is an error if the authentication method does **not** return true for

```
am.isImplicit()
```

.

Details of authenticate() method

```
Authenticate()
```

has a contract to fill in the

```
EPerson
```

of the
DSpace context with a valid ePerson when it succeeds. It gets the username either from the one passed explicitly, or by groveling credentials (such as X.509 certificates or single-sign-on credentials) out of the HTTP request environment.
It will create a
new ePerson if necessary, so long as its

```
canSelfegister()
```

allows.
It also initializes a new ePerson as used to be done by the

```
SiteAuthenticator.initEPerson
```

method.

Configuration

The authentication methods are configured as *sequence plugins* in the PluginManager. Here is a sample configuration entry, which calls on X.509, password, and the "MIT Special" methods in that order:

```
plugin.sequence.org.dspace.eperson.AuthenticationMethod = \
 org.dspace.eperson.X509Authentication, \
 org.dspace.eperson.PasswordAuthentication, \
 edu.mit.dspace.MITSpecialGroup
```

Web Service authentication

In a non-interactive context, such as Web Services, the authentication mechanism is provided with any credentials up front so there is no need for Web redirects. This simplifies the algorithm: just iterate through the stack of

```
AuthenticationMethod
```

s and try each one, stopping at the first success.

If none succeed, return the lowest-valued error result. since that is the "closest" it came to success (i.e. bad password is closer than a nonexistent user).

Comments?