User Authentication

User Authentication

Status: NOT IMPLEMENTED
Created: 14. March 2010
Author: fmeschbe
JIRA: –
References: Merging Sling API and Commons Auth API
Update: fmeschbe/27. September 2013

Update

This concept is not being implemented because in the meantime `ResourceProviderFactory` services have been introduced which can be flagged as being mandatory and thus validate credentials from authentication handlers. One such implementation is the JCR Resource Provider which does exactly that and internally validates the credentials by create a JCR Session.

Introduction

With the recent introduction of the Commons Auth Bundle and the current approach to break apart the dependency on JCR API from the Commons Auth Bundle we are faced with an issue of how to authenticate an HTTP request user while at the same time not binding the authentication mechanism to any data repository.

In other words we have the following requirements:

1. Extract user authentication information from HTTP requests and assert the identity of the requesting entity (remote user or application)
2. Setup a connection to data repository on behalf of the authenticated user

Currently the Commons Auth bundle controls the complete process of extracting authentication information, asserting the identity and connecting to the repository:

1. Authentication information extraction using `AuthenticationHandler` services
2. Asserting identity by using the authentication information to login to the JCR Repository resulting in a JCR Session.
3. Connecting to the data repository by using the `JcrResourceResolverFactory` to create a `ResourceResolver` on top of the JCR Session.

The problem here is, that the Commons Auth bundle is tied into using the JCR Repository to assert identities and into the `JcrResourceResolverFactor` to connect to the data repository.

These dependencies are not entirely optimal. So a first improvement might be for the Commons Auth bundle to validate any authentication and pass the validated authentication info on the ot Commons Auth client which then uses this data to create the connection:

1. Commons Auth extracts authentication information using `AuthenticationHandler` services
2. Commons Auth asserts the identity using the authentication information to login to the JCR Repository
3. Commons Auth returns the asserted authentication information to (say) the Sling Main Servlet which uses the `ResourceResolverFactory` to connect to the repository and return a `ResourceResolver`

The drawback here is, that (a) Commons Auth is stilled tied into the JCR API and (b) JCR Sessions are created twice thus creating quite a considerable overhead.

Proposal

A new service API is defined supporting the validation of credentials:

```
public interface CredentialValidator {

    /**
     * Validates the credentials and returns an AuthenticationInfo
     * object representing the validated credentials.
     * The implementation may return a new object or the same as
     * passed as a parameter. If a new object is returned the
     * implementation may copy some or all properties from the
     * passed in object.
     * The passed in AuthenticationInfo object should be considered
     * immutable by the implementation.
     * @param credentials The AuthenticationInfo representing the
     *      credentials provided by the user in the HTTP request.
     * @return An AuthenticationInfo object representing the
     *      validated credentials.
     * @throws LoginException if the passed credentials cannot
     *      be validated.
     * @throws NullPointerException if credentials is null
     */
    public AuthenticationInfo validate(AuthenticationInfo credentials) throws LoginException;

}
```

The `SlingAuthenticator` class makes use of the `CredentialValidator` service to validate the credentials extracted by `AuthenticationHandler` services. The returned AuthenticationInfo is then set as a request attribute.

The `CredentialValidator` interface is implemented and registered as a service by the JCR based `ResourceResolverFactory` implementation. The implementation of the method uses the credentials to authenticate with the JCR repository and returns an AuthenticationInfo object copied from the original object without the password but containing the JCR Session.

The `SlingMainServlet` gets the `AuthenticationInfo` object from the request attribute and passes it (as a `Map`) to the `ResourceResolverFactory.getResourceResolver(Map)` method to get the `ResourceResolver` for the request.

The JCR based `ResourceResolverFactory.getResourceResolver(Map)` knows about the `CredentialValidator` implementation and can make use of the `Session` object in the map to reuse the existing session.

Complete Steps Authenticating HTTP Requests

Requests are authenticated as follows:

1. Client makes HTTP Request
2. OSGi HTTP Service selects Sling to handle request and calls `HttpContext.handleSecurity`
3. Sling's `handleSecurity` method calls `SlingAuthenticator.handleSecurity`
4. SlingAuthenticator extractes `AuthenticationInfo` by calling `AuthenticationHandler.extractCredentials`
5. SlingAuthenticator passes `AuthenticationInfo` to `CredentialValidator.validate`
6. (JCR based) CredentialValidator builds JCR Credentials from `AuthenticationInfo` and calls `Repository.login`
7. CredentialValidator creates new AuthenticationInfo object copying all properties from input (except password) and setting the JCR `Session` as another property and returns
8. SlingAuthenticator sets new `AuthenticationInfo` as request attribute and sets remaining required request attributes and returns
9. Sling's `handleSecurity` returns successfully
10. OSGi HTTP Service passes control to `SlingMainServlet`
11. SlingMainServlet extracts `AuthenticationInfo` from request attribute and calls `ResourceResolverFactory.getResourceResolver` with this `AuthenticationInfo` (which actually extends `Map`)
12. (JCR based) ResourceResolverFactory recognizes the existing JCR Session and creates and returns a ResourceResolver based on this session
13. SlingMainServlet continues request processing
14. Finally SlingMainServlet closes the ResourceResolver at the end of request processing

Issues

The JCR based `CredentialValidator` implementation creates a session, which may or may not be used and closed by users of the Sling Commons Auth `AuthenticationSupport` service. A mechanism must be implemented to ensure Sessions placed into the `AuthenticationInfo` by `CredentialValidator` implementations are not left open and thus needlessly consume system resources.