Solving the Authentication Handler Credential Validation Problem

Solving the Authentication Handler Credential Validation Problem

Status: DRAFT
Created: 27. September 2013
Author: fmeschbe
JIRA: Implement solution to the Authentication Handler Credential Validation Problem, AbstractSlingRepository#login violates JCR spec
References: –
Updated: –

Problem

There does not currently exist a good and transparent way for an Authentication Handler to signal to the `ResourceResovlerFactory`, that the identity of a user has been established and validated and that no further checks are required. For example an SSO authentication handler will get the identity of a user provided by the SSO handler or an OAuth 2 authentication handler proves the identity of the user by with the help of the OAuth 2 provider.

Proposal

A new predefined property of the `AuthenticationInfo` map is defined which can be set by the authentication handler to indicate that the user's identity has been verified and can be guaranteed:

```
public interface ResourceResolverFactory {
    ....
    /**
     * Name of the authentication information property used to indicate that the
     * identity of the user indicated by the {@link #USER} property has already
     * been validated by other means such as OAuth2, OpenID or similar SSO
     * functionality. As a consequence password-less access to a
     * {@link ResourceResolver} should be granted.
     * <p>
     * The non-empty string value of this property identifies the party having
     * validated the user's identity. It may be used by implementations of this
     * and the {@link ResourceProviderFactory} interfaces in log messages.
     * <p>
     * The type of this property, if present, is <code>String</code>.
     *
     * @since 2.4 (bundle version 2.5.0)
     */
    String IDENTIFIED = "user.identified";
    ....
}
```

`ResourceProviderFactory` services creating `ResourceProvider` instances by establishing connections to the actual data store will leverage this flag to implement a pre-authentication style of access.

Implementations

Authentication Handler

Implementations will just set the `ResourceResolverFactory.IDENTIFIED` property in the Authentication Info map to the name of the authentication handler indicating the identity has been validated.

This replaces mechanisms used today such has implementing a LoginModule service validating a custom `javax.jcr.Credentials` instance.

JCR Resource Provider

The JCR Resource Provider will check for the property and create a `Subject` used for establishing the session's owner:

```
if (authenticationInfo.get("user.identified") != null) {

    // pre-identified user access
    final String userName = (String) authenticationInfo.get(ResourceResolverFactory.USER);
    final String identifier = (String) authenticationInfo.get("user.identified");

    log.info("getResourceProviderInternal: Logging in user {} identified by {}", userName, identifier);
    Session tmp = null;
    try {
        tmp = session = repository.loginAdministrative(workspace);
        Authorizable auth = ((JackrabbitSession) tmp).getUserManager().getAuthorizable(userName);
        Subject s = new Subject();
        s.getPrincipals().add(auth.getPrincipal());
        session = Subject.doAs(s, new PrivilegedExceptionAction<Session>() {
            public Session run() throws Exception {
                return repository.login(workspace);
            }
        });
    } catch (PrivilegedActionException pae) {
        throw pae.getCause();
    } finally {
        if (tmp != null) {
            tmp.logout();
        }
    }
}
```

Considerations for creating the `Subject`:

- Should the full `Subject` be created ? That is a subject which contains the user's `Principal` as well as the full set of `Principal` instances representing the set of groups of which the user is a member.
- Should only a simple `Subject` be created as in the example above ? That is only the user's `Principal` is contained and the repository implementation must then complete the set of `Principals` by the principals for the groups.
- Should a dummy `Subject` be created which only contains a simple `Principal` instance indicating the user's name (as opposed to the actual `Principal` instance representing the actual repository principal) ?
- Should a new session be retrieved for each such access or should a long-running session be used which needs to be occasionally refreshed ?
- Should mappings from user name to `Subject` be cached ? And how is that cache refreshed ?
- We must guard the use of the `user.identified` property somehow to prevent use of this feature by code to get access to other user's data (privilege escalation).

Preventing Privilege Escalation

As noted above we must make sure that no casual user can retrieve a `ResourceResolver` adding just a `user.identified` property and thus escalate his own privileges.

One approach to mitigate this problem would be to leverage the `ServiceUserMapper` service which is also used in the context of the service authentication mechanism: a sub service name `user.identified` is defined and each consumer of this mechanism must have a user mapping for this subservice to the mock user *.

This way, the JCR Resource Provider sketched above would add this check:

```
if (authenticationInfo.get("user.identified") != null) {

    if (!"*".equals(serviceUserMapper.getServiceUserID(callingBundle, "user.identified"))) {
        log.info("Missing privilege to use pre-authenticated login");
        throw new LoginException();
    }

    ...
}
```

Abstract Sling Repository

The Abstract Sling Repository implementation will need to fix the "`null` Credentials" problem: Due to a misunderstanding the Abstract Sling Repository assumes anonymous access to the repository if the `Credentials` to the login method is `null` or missing. While this has been implemented like this in Jackrabbit it is actually not foreseen by the specification. Rather missing or `null` Credentials indicate access to the repository using pre-authentication where the `Subject` identifies the owner of the `Session` to create.

So the Abstract Sling Repository implementation of the `login(String, String)` method must be fixed along these lines:

```
if (credentials == null) {
    if (Subject.getSubject(AccessController.getContext()) != null) {
        return getRepository().login(null, workspace);
    } else {
        // TODO: getAnonCredentials(this.anonUser) should not be used for anonymous access
        return getRepository().login(new GuestCredentials(), workspace);
    }
} else {
    return getRepository().login(credentials, workspace);
}
```