Java EE Container-Managed Authentication

Problem

You would like to enable Container-Managed Authentication in a Java EE web application fronted by a Shibboleth Native SP, in order to, for instance, leverage the platform's authorization capabilities, such as the EJB security model, from within your application's code. Your Application Server (AS) is, however, unaware of authentication performed by Shibboleth, effectively treating your users as *anonymous*. Thus, such API calls do not typically work right out of the box. What needs to be done is to *somehow* propagate the user identity at the SP to the AS, or in simple terms, a "caller" `Principal`[1] representing the authenticated user needs to be established there.

Solution

In order to communicate the fact that a user should be considered authenticated to your AS, you may either implement a container-specific JAAS `LoginModule`, or what is loosely termed its standard, Java EE counterpart, a JASPIC[2] `ServerAuthModule` (SAM) . This how-to employs the standardized over the proprietary approach.

Assumptions

- Your SP *passively* protects your application's resources.
- Your AS is an implementation of the Full Java EE 6+ Profile.
- Your application comprises both "protected" and "public" resources.

Implement the SAM

SAMs

SAMs closely resemble the functionality (e.g. manipulation or blocking of `HttpServletRequests` and `HttpServletResponses`) provided by common Servlet `Filters`; they may in fact be implemented as such. Unlike a `Filter`, however, a SAM is always invoked before the first `FilterChain`'s `Filter` and/or after the last one, it is used by potentially multiple applications and is capable of communicating authentication decisions to the AS.

The following example is a simplistic "bridge" SAM, which, for *every* --which is obviously not ideal-- incoming request:

1. Probes the received `HttpServletRequest` for presence of the "eduPersonPrincipalName" and "eduPersonAffiliation" attributes.
2. If both are present, registers a "caller" `Principal` and one or multiple "group" `Principals` --enclosing the values of "eduPersonPrincipalName" and "eduPersonAffiliation" as their respective names-- at the AS to represent the authenticated user.
3. Otherwise:
    a. If the targeted request is "public", it treats the user as a *guest*.
    b. Otherwise, it redirects the request to the SP's `SessionInitiator`.

Step 3.b. also occurs whenever the user explicitly asks to be authenticated, e.g. by clicking a "Login" button in the UI. Similarly, when the user wishes to be logged out, the SAM redirects her request to the SP's `LogoutInitiator`.

SAM

```
package org.my;

import java.io.IOException;
import java.security.Principal;
import java.util.Map;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.message.AuthException;
import javax.security.auth.message.AuthStatus;
import javax.security.auth.message.MessageInfo;
import javax.security.auth.message.MessagePolicy;
import javax.security.auth.message.callback.CallerPrincipalCallback;
import javax.security.auth.message.callback.GroupPrincipalCallback;
import javax.security.auth.message.module.ServerAuthModule;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
public class SpAsBridgeSam implements ServerAuthModule {

    public static class MyOrgPrincipal implements Principal {

        public MyOrgPrincipal(String name) {
            this.name = name;
        }
        private final String name;
        @Override
        public String getName() {
            return name;
        }

        }

        private static final Class<?>[] SUPPORTED_MESSAGE_TYPES = new Class<?>[] { HttpServletRequest.class,
HttpServletResponse.class };
    private static final String LOGIN_URL = "http://my.org/Shibboleth.sso/Login";
    private static final String LOGOUT_URL = "http://my.org/Shibboleth.sso/Logout";
    private boolean isProtectedResource;
    private CallbackHandler ch;

        // Post-construct initialization goes here
    @Override
    public void initialize(MessagePolicy requestPolicy, MessagePolicy responsePolicy, CallbackHandler ch, Map
options) throws AuthException {
        this.ch = ch;
            isProtectedResource = requestPolicy.isMandatory();
        }

        // Authentication logic and optional request pre-processing goes here
    @Override
    public AuthStatus validateRequest(MessageInfo mi, Subject client, Subject service) throws AuthException {
        HttpServletRequest hreq = (HttpServletRequest) mi.getRequestMessage();
        HttpServletResponse hres = (HttpServletResponse) mi.getResponseMessage();
        String username = getUsername(hreq);
        String[] groups = getUsergroups(hreq);
        if ((username != null) && (!username.trim().isEmpty()) && (groups != null)) {
            try {
                ch.handle(new Callback[] {
                        new CallerPrincipalCallback(client, new MyOrgPrincipal(username)),
                        new GroupPrincipalCallback(client, groups) });
            }
            catch (UnsupportedCallbackException | IOException e) {
                e.printStackTrace();
                throw new AuthException("Could not authenticate user.");
            }
            return AuthStatus.SUCCESS;
        }
        else if (!isProtectedResource) {
            return AuthStatus.SUCCESS;
        }
        else {
            try {
                hres.sendRedirect(LOGIN_URL);
            }
            catch (IOException ioe) {
                ioe.printStackTrace();
                throw new AuthException("Could not authenticate user.");
            }
            return AuthStatus.SEND_CONTINUE;
        }
    }

        // Validation of response and optional post-processing goes here
        // Not typically used, as --unless it had initially been wrapped-- the response is already committed
        // by the time this method is called
    @Override
    public AuthStatus secureResponse(MessageInfo messageInfo, Subject service) throws AuthException {
```

```
        return AuthStatus.SEND_SUCCESS;
    }

        // Logout logic goes here
        // The spec is a bit unclear regarding this method, thus I'm unsure whether it's best to directly
modify the Subject
        // or use container facilities
        @Override
    public void cleanSubject(MessageInfo mi, Subject client) throws AuthException {
        HttpServletRequest hreq = (HttpServletRequest) mi.getRequestMessage();
        HttpServletResponse hres = (HttpServletResponse) mi.getResponseMessage();
        HttpSession hs = hreq.getSession(false);
        if (hs != null) {
            try {
                hs.invalidate();
            }
            catch (IllegalStateException ise) {
                ise.printStackTrace();
                throw new AuthException("Could not invalidate user session.");
            }
        }
        try {
            hreq.logout();
            hres.sendRedirect(LOGOUT_URL);
        }
        catch (ServletException | IOException e) {
            e.printStackTrace();
            throw new AuthException("Could not complete user logout.");
        }
    }

    @Override
    public Class<?>[] getSupportedMessageTypes() {
        return SUPPORTED_MESSAGE_TYPES;
    }

    private String getUsername(HttpServletRequest hreq) {
        return (String) hreq.getAttribute("eduPersonPrincipalName");
    }

    private String[] getUsergroups(HttpServletRequest hreq) {
        String groupsAttribute = (String) hreq.getAttribute("eduPersonAffiliation");
        String[] groups = null;
        if (groupsAttribute != null) {
            groups = groupsAttribute.split(";");
            for (int i = 0; i < groups.length; i++) {
                groups[i] = groups[i].trim();
            }
        }
        return groups;
    }

}
```

Deploy the SAM

SAMs are typically deployed autonomously on the AS, at a location monitored by the common class loader; you may however be able to deploy yours as part of your application. As, unfortunately, this step is vendor-specific, you will have to consult the documentation of your AS to complete it. On GlassFish 4+, directories where SAMs can be placed are by default `<as-install>/lib` and <as-install>/`domains/<domain_name>/lib`.

Here is an example GlassFish `domain.xml` snippet:

<as-install>/domains/<domain_name>/config/domain.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<domain log-root="${com.sun.aas.instanceRoot}/logs" application-root="${com.sun.aas.instanceRoot}/applications"
version="13">
...
        <configs>
        ...
            <config name="server-config">
                        ...
                        <security-service>
                                ...
                                <message-security-config auth-layer="HttpServlet">
                                        ...
                                        <!-- "provider-type" attribute is typically "server" -->
                                        <provider-config provider-type="server" provider-id="SpAsBridgeSam"
class-name="org.my.SpAsBridgeSam">
                                                <!-- Options that will be passed to your SAM via the "options"
Map argument of its initialize method -->
                                                <property name="org.my.SpAsBridgeSam.FOO" value="BAR"/>
                                                ...
                                                <request-policy/>
                                                <response-policy/>
                                        </provider-config>
                                        ...
                                </message-security-config>
                                ...
                        </security-service>
                        ...
                </config>
                ...
        </configs>
        ...
</domain>
```

Declare Application Roles

Now you will have to declare your application's roles, by annotating types or placing appropriate elements within the various Deployment Descriptors (DD), e.g. for a web module:

WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    id="WebApp_ID"
        version="3.1"
        metadata-complete="false"
>
        ...
    <security-constraint>
            <web-resource-collection>
            <web-resource-name>myOrgProtectedResources</web-resource-name>
            <url-pattern>/protected/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>staff</role-name>
                        <role-name>student</role-name>
        </auth-constraint>
    </security-constraint>
    <security-role>
        <role-name>staff</role-name>
                <role-name>student</role-name>
    </security-role>
</web-app>
```

Map SAM Groups to Application Roles

This final step is once again a vendor-specific (and somewhat frustrating) one. You will have to provide --via a proprietary DD or your product's administration console / cli tools-- a mapping between groups returned by the SAM and the Java EE roles used by your application (even if they are the same, as they commonly are).

Here is an example GlassFish 4 `glassfish-web.xml snippet`:

WEB-INF/glassfish-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app        PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Servlet 3.0
//EN" "http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app httpservlet-security-provider="SpAsBridgeSam">
    <context-root>/myApp</context-root>
    <security-role-mapping>
        <role-name>staff</role-name>
        <group-name>staff</group-name>
    </security-role-mapping>
        <security-role-mapping>
                <role-name>student</role-name>
                <group-name>student</group-name>
        </security-role-mapping>
</glassfish-web-app>
```

Restart AS, Deploy Application

Now What?

You should now be able to leverage Java EE declarative or programmatic authorization from within your code, e.g.:

```
@Stateless
public class SomeService {

        @RolesAllowed("staff")
        public void deleteEverything() {
                ...
        }

}
```

...inject `Principal`s into your beans via CDI, use methods like `authenticate()` and `isCallerInRole()`, or, if you are brave, venture into the arcane world of JACC.

---

1 Java EE mostly disregards `javax.security.auth.Subject`s; all what's left of them, that is, what most Java EE specifications tend to use, is this one "caller" `Principal`, along with potential "group" `Principal`s. A blog entry by Arjan Tijms titled "JAAS in Java EE is not the universal standard you may think it is" examines some common misconceptions surrounding JAAS in this scope.

2 JSR 196, Authentication Service Provider Interface for Containers (JASPIC, JASPI) is available in the Full Java EE 6+ Profile, and Servlet 3+ Containers included in compatible implementations have been mandated to support JASPIC's Servlet Profile.