Prev Next

Chapter 14. Creating a Post Authentication Plugin

Table of Contents

Post authentication plugins (PAP) let you include custom processing at the end of the authentication process, immediately before the subject is authenticated. Common uses of post authentication plugins include setting cookies and session variables. Post authentication plugins are often used in conjunction with policy agents. The post authentication plugin sets custom session properties, and then the policy agent injects the custom properties into the request header to the protected application.

This chapter explains how to create a post authentication plugin.

14.1. Designing Your Post Authentication Plugin

Your post authentication plugin class implements the `AMPostAuthProcessInterface` interface, and in particular the following three methods.

```
public void onLoginSuccess(
  Map requestParamsMap,
  HttpServletRequest request,
  HttpServletResponse response,
  SSOToken token
) throws AuthenticationException

public void onLoginFailure(
  Map requestParamsMap,
  HttpServletRequest request,
  HttpServletResponse response
) throws AuthenticationException

public void onLogout(
  HttpServletRequest request,
  HttpServletResponse response,
  SSOToken token
) throws AuthenticationException
```

OpenAM calls the `onLoginSuccess()` and `onLoginFailure()` methods immediately before informing the user of login success or failure, respectively. OpenAM calls the `onLogout()` method only when the user actively logs out, not when a user's session times out.

See the *OpenAM Java SDK API Specification* for reference.

These methods can perform whatever processing you require. Yet, know that OpenAM calls your methods synchronously as part of the authentication process. Therefore, if your methods take a long time to

complete, you will keep users waiting. Minimize the processing done in your post authentication methods.

14.2. Building Your Sample Post Authentication Plugin

v11.0.1 available

The following example post authentication plugin sets a session property during successful login, writing its debug log if the operation fails.

```
package com.forgerock.openam.examples;

import java.util.Map;

import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;

import com.sun.identity.authentication.spi.AMPostAuthProcessInterface;
import com.sun.identity.authentication.spi.AuthenticationException;
import com.sun.identity.shared.debug.Debug;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SamplePAP implements AMPostAuthProcessInterface
{
  private final static String PROP_NAME = "MyProperty";
  private final static String PROP_VALUE = "MyValue";
  private final static String DEBUG_FILE = "SamplePAP";

  protected Debug debug = Debug.getInstance(DEBUG_FILE);

  public void onLoginSuccess(
    Map requestParamsMap,
    HttpServletRequest request,
    HttpServletResponse response,
    SSOToken token
  ) throws AuthenticationException
  {
    try {
      token.setProperty(PROP_NAME, PROP_VALUE);
    } catch (SSOException ssoe) {
      debug.error("Unable to set property");
    }
  }

  public void onLoginFailure(
    Map requestParamsMap,
    HttpServletRequest request,
    HttpServletResponse response
  ) throws AuthenticationException
  {
    ; // Not used
  }

  public void onLogout(
    HttpServletRequest request,
    HttpServletResponse response,
    SSOToken token
  ) throws AuthenticationException
  {
    ; // Not used
  }
}
```

A post authentication plugin code relies on three .jar files, two of which are deployed with OpenAM, and the third which is provided by your web application container.

`amserver.jar`

When you deploy OpenAM, the file is *`war-file-name`*`/WEB-INF/lib/amserver.jar`.

`opensso-sharedlib.jar`

When you deploy OpenAM, the file is *`war-file-name`*`/WEB-INF/lib/opensso-sharedlib.jar`.

`servlet-api.jar`

This .jar provides the Java EE Servlet API.

If you use Apache Tomcat as your web application container, the file is `/path/to/tomcat/lib/servlet-api.jar`.

Put the sample plugin in `src/com/forgerock/openam/examples/SamplePAP.java`, and compile the class.

```
$ cd src
$ mkdir ../classes
$ javac
 -d ../classes
 -classpath /path/to/tomcat/webapps/openam/WEB-INF/lib/amserver.jar:
 /path/to/tomcat/webapps/openam/WEB-INF/lib/opensso-sharedlib.jar:
 /path/to/tomcat/lib/servlet-api.jar
 com/forgerock/openam/examples/SamplePAP.java
```

Copy the classes to the `WEB-INF/classes` directory where you deployed OpenAM.

```
$ cp -r ../classes/* /path/to/tomcat/webapps/openam/WEB-INF/classes/
```

Restart OpenAM or your web container to ensure the post authentication plugin class is loaded.

```
$ /etc/init.d/tomcat stop
$ /etc/init.d/tomcat start
$ tail -1 /path/to/tomcat/logs/catalina.out
INFO: Server startup in 32070 ms
```

14.3. Configuring Your Post Authentication Plugin

You can configure the post authentication plugin for a realm, for a service (authentication chain), or for a role. Where you configure the plugin depends on the scope to which the plugin should apply. Configuring the plugin at the realm level as shown here, for example, ensures that OpenAM calls your plugin for all authentications to the realm.

In the OpenAM console, browse to Access Control > *`Realm Name`* > Authentication > All Core Settings... In the Authentication Post Processing Classes list, add the sample plugin class, `com.forgerock.openam.examples.SamplePAP`, and then click Save.

Alternatively, you can configure sample plugin for the realm by using the `ssoadm` command.

```
$ ssoadm
```

```
 set-svc-attrs
 --adminid amadmin
 --password-file /tmp/pwd.txt
 --servicename iPlanetAMAuthService
 --realm /realm
 --attributevalues iplanet-am-auth-post-login-process-class=
 com.forgerock.openam.examples.SamplePAP

iPlanetAMAuthService under /realm was modified.
```


14.4. Testing Your Post Authentication Plugin

To test the sample post authentication plugin, login successfully to OpenAM in the scope where the plugin is configured. For example, if you configured your plugin for the realm, `/realm`, specify the realm in the login URL.

```
http://openam.example.com:8080/openam/UI/Login?realm=realm
```

Although as a user you do not notice anywhere in the user interface that OpenAM calls your plugin, a policy agent or custom client code could retrieve the session property your plugin added to the user session.

Prev *Next*
Chapter 13. Customizing Authentication Modules *Home* *Chapter 15. Customizing Policy Evaluation*