Impersonation Authentication module for OpenAM

- Introduction
- Configuration
- Development
  - Configuration read from Module Instance
  - Authorize the administrator locally
  - Authorize the Administrator
  - Parse the JSON response from Policy Evaluation
- Demo
- Source

Introduction

Support for impersonation is useful in the enterprise use cases where designated administrators are required to act on behalf of a user in certain scenarios. By impersonating another user as an administrator, if authorized to do so, gains access to a restricted view of the user’s profile in the system. This is helpful in situations involving password reset, request-based access and profile updates. However, the design of such a system must call for controls that actively restrict access to the user’s entitlements at the outset. This can be achieved using step-up authentication for gaining access to private user data, and also by using the OpenAM policy engine for performing advanced resource-based decisioning.

Configuration

An OpenAM custom authentication module was written to enable impersonation support. The module requires input of the username of the end-user being impersonated and the administrator credentials. After submitting the username and password, the admin account is authenticated first and then it is also authorized to complete the impersonation request using REST calls to a specified OpenAM Policy endpoint. This policy can be either local or external as we shall examine further. The impersonated user is also validated for being in active status in the system. If all is okay, the administrator is permitted to impersonate and OpenAM creates a session for the impersonated user. The module can be configured using the following gauges to complete the described functions correctly:

Impersonation

<table>
<thead>
<tr>
<th>Realm Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authentication Level:</strong></td>
</tr>
<tr>
<td>The authentication level associated with this module.</td>
</tr>
<tr>
<td><strong>Group Attribute Name:</strong></td>
</tr>
<tr>
<td>Name of the attribute where response is stored</td>
</tr>
<tr>
<td><strong>Impersonation User Id:</strong></td>
</tr>
<tr>
<td>The user is prompted for the user id</td>
</tr>
<tr>
<td><strong>Resource Set:</strong></td>
</tr>
<tr>
<td>The resource set against which the module evaluates policies</td>
</tr>
<tr>
<td><strong>Policy Realm:</strong></td>
</tr>
<tr>
<td>The policy container</td>
</tr>
<tr>
<td><strong>Authentication Realm:</strong></td>
</tr>
<tr>
<td>The realm to authenticate the administrator</td>
</tr>
<tr>
<td><strong>Policy Set or Application name:</strong></td>
</tr>
<tr>
<td>The policy set or application name</td>
</tr>
<tr>
<td><strong>Check group membership:</strong></td>
</tr>
<tr>
<td>Check group membership</td>
</tr>
<tr>
<td><strong>OpenAM Server:</strong></td>
</tr>
<tr>
<td>OpenAM Server</td>
</tr>
</tbody>
</table>

1. Setup the resource-set you want to check policy for. This resource set its nothing but a special URL that invokes policy evaluation for impersonation
2. The authentication realm you want the administrator to authenticate in. The authentication module allows for realm-specific authentication
3. The OpenAM server where the policy resides, the realm where the policy resides, and the policy-set name. The policy does not need to be local and can be on a remote policy host
4. Check whether you want the administrator to be a member of a local group as well, in addition to the external policy authorization.

A step by step account of the workings of the module follows.
Configuration read from Module Instance


Authorize the administrator locally

In our test scenario, the 'user.0' is really an administrative user that has been granted membership to the group named 'impersonation', as configured in the module (see above).

We build an AMIdentity object for the group and validate membership.

[AMIdentity object: id=impersonation, ou=group, o=impersonation, ou=services, dc=openam, dc=forgerock, dc=org]
value of attribute: uid=user.0, ou=People, dc=forgerock, dc=com

userName to check: user.0
match found! admin: user.0 allowed to impersonate user: user.1

Authorize the Administrator

Get the ssotoken for the admin who is trying to impersonate via a policy call, and authenticate the user to the realm specified in the config

Get the ssotoken for the admin who is trying to impersonate via a policy call, and authenticate the user to the realm specified in the config

tokenId-> AQIC5wM2LY4Sfcxokjdayf3ig00DuQIXT9T98_3h9q72A.*AAJTSQACMDEEA1NLAB10DK0Ng1NTExNDUzNzcxNDI.*

Build the 2nd policy rest call, and use the resource set, openam server, policy set and policy container from the configuration passed to the module.

stringentity->{"resources": ["http://openam:8080/openam/index.html"], "application": "impersonation", "subject": {"ssToken": "AQIC5wM2LY4Sfcxokjdayf3ig00DuQIXT9T98_3h9q72A.*AAJTSQACMDEEA1NLAB10DK0Ng1NTExNDUzNzcxNDI.*"}}


Custom response attributes can be passed back to the module for further evaluation if needed. For example, a statically defined roleName-timeBoundAdmin could be used to further restrict this impersonation request within the time window specified in the 'timeBoundAdmin' control. This example is only given to seed the imagination, the module currently does not restrict the impersonation session using a time window, but this is possible to do.

Parse the JSON response from Policy Evaluation


If the ACTION set returned for GET/POST is TRUE, the admin is permitted to impersonate. This could be extended to include other actions as necessary. Finally, destroy the admin session, now that it is not needed anymore and return the impersonated user as the Principal for constructing an OpenAM session.

Demo

Our short demo begins with the administrator being asked for the username of the user they want to impersonate.
Next, the module asks for the admin credentials.

**User authentication needed**

User Name: user.1
Password: ********

If the administrator is unable to authenticate, or does not belong to the local group, or fails external policy evaluation, the following error screen is shown:

⚠️ You are not authorized to impersonate!

Contact your system administrator.

If all checks pass, the administrator is granted the user's session and logs into OpenAM.

Source
https://github.com/javedmshah/impersonation-policy