Writing a controller for a secured page

Concepts

A secured page

A secured page in VIVO is one that can not be viewed by the general public. If an unauthorized user attempts to view a secured page, even by entering the URL directly into a browser, the attempt should fail.

How is a page secured?

To secure a page, the controller code requests authorization to perform a particular `RequestedAction`. If the user is not authorized to perform that action, the controller rejects the request. For example, the `RevisionInfoController` checks to see whether the user is authorized for the `SEE_REVISION_INFO.ACTION`. If the user is not authorized for that action, they will not see the Revision Info page.

Other controllers use more complicated tests to determine whether a user is authorized. For example, the `ManageProxiesAjaxController` permits access by any user who is authorized for either the `MANAGE_PROXIES.ACTION` or the `MANAGE_OWN_PROXIES.ACTION`.

Who may view a secured page?

A secured page can never be viewed by someone who is not logged in to VIVO. Since we don't know who the user is, we can't know whether they are authorized to view the page.

If a user is logged in, there is a list of `Identifiers` associated with their account. The `Identifiers` are pieces of information about that user, including their account URI, the URI of their profile page, their assigned role, any proxy permissions, and more. When a secured page is requested, these `Identifiers` are passed to the list of active `Policy` objects. Each `Policy` applies its own logic to determine whether the user may view the secured page.

What happens if the user is not authorized?

- If the user is logged in, but does not have authorization to view the secured page, the browser will be redirected to the VIVO home page. A message at the top of the page will state that the user is not authorized to view the page he requested.
- If the user is not logged in, the browser will be redirected to the VIVO login page. When the user logs in, the browser will be redirected to the secured page, and the test is repeated.
  - If the user is authorized, the secured page will be displayed.
  - If the user is not authorized, the home page will be displayed, as previously described.

What happens when a user logs out?

If a user is viewing an unsecured page, and clicks on the "Log out" link, the page will be refreshed. For some pages, particularly profile pages, the contents of the page may have changed. Many people appreciate this feature when editing their own profiles. Log in, and you can edit. Log out, and you can see what your page looks like to the public.

If a user is viewing a secured page and clicks on the "Log out" link, the browser will be redirected to the VIVO home page.

Requested Actions

Requested actions are usually quite simple. For example, the `RevisionInfoController` requests permission to display the revision info page. The user either has that permission or they do not.

On the other hand, Requested actions can be quite detailed. For example, the `ImageUploadController` requests permission to add or modify a particular triple in the data model. If the user is logged in as root or admin, they have permission. However, if the user is logged in as a self-editor, a complex algorithm is performed to determine whether they are authorized to add or modify the triple in question. They may be authorized because the subject of the triple is the URI of their own profile page, or because they have been given proxy rights to edit the page in question, or several other possibilities.

The most common case

The most common scenario for a secured page is when a simple, unparameterized action is requested, and the user either

- has a permission that provides authorization, or
- does not have that permission and is not authorized.

The steps

Decide on a permission and requested action

Simple permissions like this are usually implemented by the `SimplePermission` class, which also provides an implementation for the corresponding `RequestedAction`.

In some cases, it makes sense to re-use an existing instance of `SimplePermission`. So for example, `SimplePermission.USE_ADVANCED_DATA_TOOLS_PAGES` authorizes the user for any and all of the RDF ingest/export pages. In other cases, it makes more sense to create a new instance. So `SimplePermission.MANAGE_PROXIES` stands alone with only one usage.

For this example, we will look at `SimplePermission.SEE_REVISION_INFO`, which has only one usage.

Write the controller to require the requested action

If the controller extends `FreemarkerHttpServlet`, override the `requiredActions()` method, like this:

```
@Override
protected Actions requiredActions(VitroRequest vreq) {
    return SimplePermission.SEE_REVISION_INFO.ACTIONS;
}
```

If the controller exends `VitroHttpServlet` (but not `FreemarkerHttpServlet`), add a test to the `doGet()` and `doPost()` methods, like this:

```
@Override
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
    if (!isAuthorizedToDisplayPage(req, resp,
            SimplePermission.SEE_REVISION_INFO.ACTIONS)) {
        return;
    }
...
```

Both of these examples take advantage of the fact that each instance of `SimplePermission` defines its own `RequestedAction`, as well as its own `Actions` set.

Grant the permission to the desired users.

Each `Permission`, simple or otherwise, can be assigned to `PermissionSets` within VIVO. Each user account is associated with a `PermissionSet`, and may use the `Permissions` associated with it. The assignment of `Permissions` to `PermissionSets` occurs in the file called

```
[vitro]/webapp/rdf/auth/everytime/permission_config.n3
```

By inspecting the RDF in this file, we can see that the `SEE_REVISION_INFO` permission is assigned to `ADMIN`, `CURATOR`, and `EDITOR` `PermissionSets`. Here is an excerpt of the file with the relevent RDF:

```
@prefix auth: <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#> .
@prefix simplePermission: <java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#> .

auth:ADMIN auth:hasPermission simplePermission:SeeRevisionInfo .
auth:CURATOR auth:hasPermission simplePermission:SeeRevisionInfo .
auth:ADMIN auth:hasPermission simplePermission:SeeRevisionInfo .
```

In future versions of VIVO, the `Permission/PermissionSet` framework may be extended to permit multiple `PermissionSets` per user, with GUI-based configuration.

A more complex example

TBD

- *It's all about the action that your controller is requesting, and whether your user has authorization to do it.*
  - *Actions can be parameterized (modify this statement) or not (see the revision info page)*
  - *Authorization can come from a policy, or from a permission*
  - *Permissions can be simple, or as complex as a policy*
- *Look at the simplest case: RevisionInfoController*
  - *Not parameterized: SimplePermission.something.ACTION*
- *Code in HttpServlet, FreemarkerServlet, JSP*
- *Look at a complex case: ImageUploadController*
  - *Also ManageProxiesAjaxController*
- *In some cases, it isn't a question of whether your controller will run, but what it will do:*
  - *BaseIndividualTemplateModel*
  - *public boolean isEditable() {*

    AddDataPropertyStatement adps = new AddDataPropertyStatement(

    vreq.getJenaOntModel(), individual.getURI(),

    RequestActionConstants.SOME_URI);

    AddObjectPropertyStatement aops = new AddObjectPropertyStatement(

    vreq.getJenaOntModel(), individual.getURI(),

    RequestActionConstants.SOME_URI,

    RequestActionConstants.SOME_URI);

    return PolicyHelper.isAuthorizedForActions(vreq, new Actions(adps).or(aops));

    }
  - *LoginRedirector*

    - *private boolean canSeeSiteAdminPage() {*

      return PolicyHelper.isAuthorizedForActions(request,

      SimplePermission.SEE_SITE_ADMIN_PAGE.ACTIONS);

      }
  - *BaseSiteAdminController*
    - *if (PolicyHelper.isAuthorizedForActions(vreq, SimplePermission.MANAGE_USER_ACCOUNTS.ACTIONS)) {*

      data.put("userAccounts", UrlBuilder.getUrl("/accountsAdmin"));

      }