Main document describing authentication/authorization in CRIC

Authorization implementation

from Alexey's mail

I've implemented CRIC Authorization architecture and in particular the authorization via CERN SSO shibboleth that will provide convenient way to allow access and restrict user actions within the system. For the moment, following key building blocks have been implemented which I believe should cover well all our authorization use-cases: Overall cric_auth application that implements proper User related models and provides functional way to co-exist several authentication methods together through appropriate backends and middlewares against various sources in the system. An examples of such sources: central CERN SSO, VOMS, locally managed SSL-based authentication, locally managed password-based authentication, etc,, Extended User permissions model included support of 2 essential features: "standard" permissions plus instance-specific permissions/restrictions for given object or set of objects Extended Group model to keep the source of privileges Unified way of Model restrictions definition Unified permissions check via decorators to be applied for given view in the system Unified login views (aggregating all available auth backend methods) to let end user select appropriate method of authentication (for the moment SSO & local password-based authentication are ACTIVATED) Authorization via SSO: ssoauth athorization module and corresponding User profile to keep external user information from central CERN db Authorization via username/passwd (adopted default Django authorization) Integration with built-in Django admin panel maybe something else, but I did not remember already:)

Just a short summary about what kind of User permissions we are able to manage for today: Global permissions - a permission which is not attached to a Model of Information Schema (IS), it's a general way to implement user's allowed actions, for example, can_update_something, can_edit_sensitive_form_settings Model permissions - a permission attached to a Model of IS, predefined set of permissions: can create/update/delete all objects for given Model, e.g. for Site: can create Site objects Instance-specific permission: a model permission with restrictions for particular instance of the Model. e.g. for Site: can update ONLY Sites from CH country or can update only site with name=CERN-PROD Real examples of permissions names:

model permission: "core.change_experimentsite" instance specific: "core.change_experimentsite;vo=atlas" or "core.change_experimentsite;country=Switzerland;vo=atlas" or "core.change_experimentsite;country=Switzerlan;vo=atlas" or "core.change_experimentsite;rcsite=CERN-PROD"

As an example I deployed updates to the CRIC instance (compass oriented) at aiatlas175.cern.ch https://aiatlas175.cern.ch http://aiatlas175.cern.ch/accounts/login/

Everyone from cric-dev authorized via SSO will be automatically granted admin privileges in the system. Everyone is welcome to check at least unified login page http://aiatlas175.cern.ch/accounts/login/ ..

Some technical details:

1. An example of view where illustrated how to apply the permissions checks (via decorators): https://gitlab.cern.ch/cric/cric/blob/master/lib/cric/apps/core_web/views/site_views.py#L87

@method_decorator(login_required) @method_decorator(permission_required(['core.change_site'], Site)) @transaction.atomic def op__edit(self, request, **kwargs):

2. An example how to apply instance specific permissions check: https://gitlab.cern.ch/cric/cric/blob/master/lib/cric/apps/core_web/views/site_views.py#L100

required_permissions = ['core.change_site'] if not request.user.has_perms(required_permissions, (obj, ['rcsite', 'country'])): raise PermissionDenied(required_permissions, 'edit', obj)

3. An example how to define at the level of Model the list of supported restrictions which can be translated to Model relations/attributes: https://gitlab.cern.ch/cric/cric/blob/master/lib/cric/apps/core/models/site.py#L63

class Site(models.Model, RestrictedInstance):

ALLOWED_RESTRICTIONS = { 'country': lambda obj: obj.country.name, 'rcsite': lambda obj: obj.name, }

4. In case of complicated relations or custom logic, a model has to implement resolve_restrictions() function, the default of implementation of which is following
https://gitlab.cern.ch/cric/cric/blob/master/lib/cric/apps/core/models/base.py#L16

Current development and issues

- VOMS authentication/authorization implementation is followed up in this JIRA ticket

-- JuliaAndreeva - 2017-08-28

This topic: LCG > CRICAuthenticationAuthorization
Topic revision: r2 - 2017-08-28 - JuliaAndreeva