External authentication for Django projects

Jan Pazdziora
Senior Principal Software Engineer
Identity Management Engineering, Red Hat
jpazdziora@redhat.com

EuroPython
24th July 2015

Situation

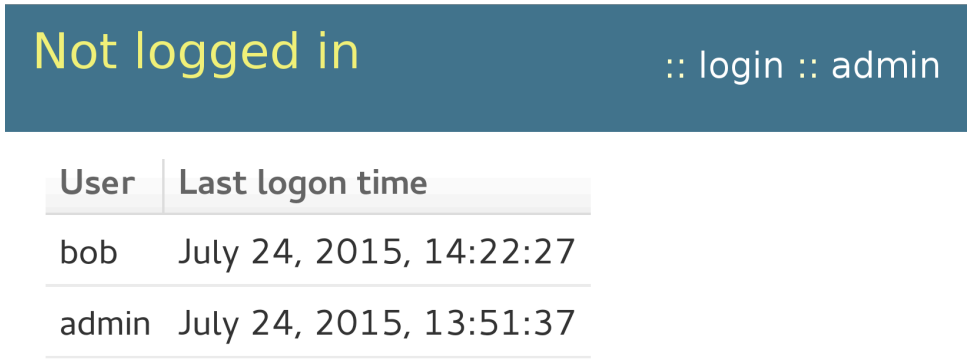
- You've developed successful application using Django.
- It makes use of users, presumably based on `django.contrib.auth`.
- Now a large organization plans to deploy the application.
  - All its employees should have access.
  - Plus associates of its partners and suppliers are to use it.
- Desired workflow:
  - New person joins the organization.
  - They can immediatelly start using the application.
  - Preferrably using single sign-on (SSO).
  - Authorization derived from group membership in the central system.

Problem statement

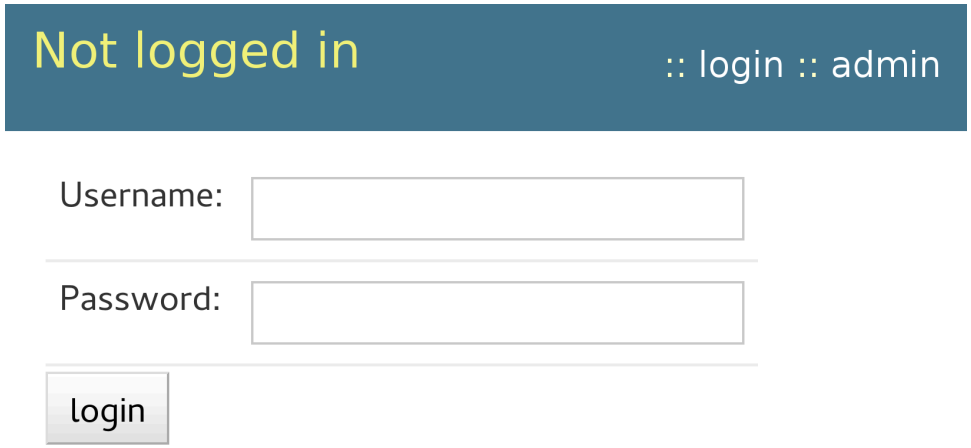
- Large organizations have existing identity management solutions.
  - FreeIPA/IdM, Active Directory, LDAP servers, ...
  - With user groups used for access control.
- Admins will not create nor manage users in the application manually.
- Organizations use standard authentication mechanisms and setups:
  - Kerberos / GSSAPI
  - Access cards / X.509 / SSL client authentication
  - SAML
- Organizations often mandate that authentication is done in their verified frontend setup.

Simple setup

- Assume the application uses `django.contrib.auth`.



- With `django.contrib.auth.views.login` and some custom template.



Authentication in frontend HTTP server

- We will look at Apache with mod_wsgi but the story is generic.
- AuthType set up in Apache configuration.
- It sets REMOTE_USER.
- Easy answer:

```
MIDDLEWARE_CLASSES = [
    ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.RemoteUserMiddleware',
    ...
]
AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.RemoteUserBackend',
]
```

- Or is it?

Limits of `RemoteUserMiddleware`

- It assumes external (Apache) authentication covers all locations/URLs that are to be seen as authenticated by the application.
  - Apache would need to authenticate every request.
    - Or maintain authentication-related sessions.
  - You do not want to renegotiate Kerberos upon every HTTP request.
  - We need Apache authentication on `/login` URL(s) only.
- The `django.contrib.auth.views.login` does not understand when `RemoteUserMiddleware` has already authenticated the user.
  - It will still show the login form.
  - Even if the user is authenticated for the request.

Authentication on single URL

- Externally-authenticated login URL should initiate authenticated Django session.

```
<Location /login/>
  AuthType GSSAPI
  AuthName "Kerberos Login"
  GssapiCredStore keytab:/etc/http.keytab
  # GssapiLocalName on
  # Require valid-user
  Require pam-account fin-app-prod
</Location>
```

```
<Location /login/>
  SSLVerifyClient require
</Location>
```

```
<Location /login/>
  MellonEnable "auth"
</Location>
```

- This way it is easier to fall back to the application-provided login mechanism if needed, making the external authentication optional.

Make it persistent

- New `PersistentRemoteUserMiddleware` in Django 1.9.
- A drop-in replacement for `RemoteUserMiddleware`:

```
MIDDLEWARE_CLASSES = [
    ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.PersistentRemoteUserMiddleware',
    ...
]
```

- It keeps the user authenticated.

External authentication aware login

- Wrap `django.contrib.auth.views.login` with code actively checking `request.user.is_authenticated()`:

```
from django.contrib.auth.views import login as auth_login
# additional imports

def login(request, template_name='activity/login.html',
        redirect_field_name=REDIRECT_FIELD_NAME):
    if hasattr(request, 'user') and request.user.is_authenticated():
        redirect_to = request.POST.get(redirect_field_name,
            request.GET.get(redirect_field_name, ''))
        if not is_safe_url(url=redirect_to, host=request.get_host()):
            redirect_to = resolve_url(settings.LOGIN_REDIRECT_URL)
        return HttpResponseRedirect(redirect_to)
    return auth_login(request, template_name = template_name,
        redirect_field_name = redirect_field_name)
```

- Have you got idea for better solution?
  - Chime in in ticket # 25164.

Additional user attributes

- With external authentication, traditionally only the login name is used, provided in `REMOTE_USER`.
- Modern Web applications want to send emails to their users.
- Using "<username>@<application's domain>" often does not work.
- Modern Web applications would like to make the UI nice by knowing user's name.
- And other attributes.
- Let's introduce `REMOTE_USER_<attribute>` variables.

Populating REMOTE_USER_<attribute>

- For SSSD-based installations, mod_lookup_identity can be used:

```
<Location /login/>
  LookupUserAttr mail REMOTE_USER_EMAIL
  LookupUserAttr givenname REMOTE_USER_FIRSTNAME
  LookupUserAttr sn REMOTE_USER_LASTNAME
</Location>
```

  Note: remapping in `ldap_user_extra_attrs` could also be used.

- For SAML, the mod_auth_mellon module can populate the attributes from <saml:AttributeStatement>:

```
<Location /login/>
  MellonSetEnvNoPrefix REMOTE_USER_EMAIL email
  MellonSetEnvNoPrefix REMOTE_USER_FIRSTNAME givenname
  MellonSetEnvNoPrefix REMOTE_USER_LASTNAME surname
</Location>
```

Consuming attributes in Django

```
# The real code has a few more checks
class RemoteUserAttrMiddleware(RemoteUserMiddleware):
    def process_request(self, request):
        if hasattr(request, 'user') and request.user.is_authenticated() \
          and user.get_username() == request.META[self.header]:
            stored_backend = load_backend(request.session.get(BACKEND_SESSION
            if isinstance(stored_backend, RemoteUserBackend):
                email = request.META.get("REMOTE_USER_EMAIL", None)
                if email is not None:
                    request.user.email = email
                firstname = request.META.get("REMOTE_USER_FIRSTNAME", None)
                if firstname is not None:
                    request.user.first_name = firstname
                lastname = request.META.get("REMOTE_USER_LASTNAME", None)
                if lastname is not None:
                    request.user.last_name = lastname
                request.user.save()
```

- Upon every login, user in Django's `auth_user` table gets synchronized.

Group-based authorization

- Many organizations diligently manage user group membership in their central identity management system.
- They want to be able to assign application-level permissions to groups.
- And have group memberships propagated without manual edits.
- Plan:
  - Upon user login, propagate their external group membership into Django groups starting with `ext:` prefix.
  - Application admins will create `ext:`-prefixed groups for groups that are relevant for the application and assign permissions to them.
  - Nonprefixed group are available for local group management.

Populating REMOTE_USER_GROUP_*

- For SSSD-based installations, mod_lookup_identity can be used:

```
<Location /login/>
  ...
  LookupUserGroupIter REMOTE_USER_GROUP
</Location>
```

- The mod_auth_mellon module can populate attributes from SAML response:

```
<Location /login/>
  ...
  MellonEnvVarsSetCount On
  MellonEnvVarsIndexStart 1
  MellonSetEnvNoPrefix REMOTE_USER_GROUP groups
</Location>
```

Example result:

```
REMOTE_USER_GROUP_N=2
REMOTE_USER_GROUP_1=network-admin-emea
REMOTE_USER_GROUP_2=network-admin-na
```

Consuming external groups

```
# extending RemoteUserAttrMiddleware
class RemoteUserAttrMiddleware(RemoteUserMiddleware):
    group_prefix = 'ext:'

    def update_user_groups(self, request):
        user = request.user
        ext_group_count = request.META.get("REMOTE_USER_GROUP_N", None)
        current_groups = {}
        for g in user.groups.filter(name__startswith=self.group_prefix):
            current_groups[g.name] = g
        for i in range(1, int(ext_group_count) + 1):
            if request.META.get("REMOTE_USER_GROUP_" + str(i), None):
                g = self.group_prefix + request.META["REMOTE_USER_GROUP_" + s
                if current_groups.has_key(g):
                    del current_groups[g]
                else:
                    g_obj = Group.objects.filter(name=g)
                    if g_obj:
                        user.groups.add(g_obj[0])
        for g in current_groups.values():
            user.groups.remove(g.id)
```

Consuming external groups (cont'd)

```
    def process_request(self, request):
        ...
                self.update_user_groups(request)
                request.user.save()
```

Conclusion

- It is possible to support wild combination of authentication methods, by using authentication frontends.
- With `PersistentRemoteUserMiddleware`, isolated login URLs work.
- The `login` methods need to be checked and possibly amended to observe the external authentication.
- Merely login name in `REMOTE_USER` is often not sufficient.
- With custom middleware like `RemoteUserAttrMiddleware`, user attributes and group memberships can stay in sync.
- When new associate logs in, they can not just do it via SSO, they will have their account and permissions fully set up, automatically.
- No Python code specific to the authentication methods was written.
- This is call for comments: do you find the approach useful?

References

- www.freeipa.org/page/Environment_Variables#Proposed_Additional_Variables
- www.freeipa.org/page/Web_App_Authentication
- code.djangoproject.com/ticket/25164
- www.adelton.com/django/