django-valet-keys Documentation

Release 0.0.1

Les Orchard

January 16, 2015

Contents

django-valet-keys is a Django app for managing valet keys to let robots drive around in your identity.

- Build status on travis-ci ()
- Latest documentation on Read The Docs (source)
- Task board on Huboard (source)

Contents:

CHAPTER 1

Overview

1.1 What is django-valet-keys?

There are many ways to allow a robot to authenticate to a web service and perform actions on your behalf. For example:

- HTTP Basic authentication using your user name and password
- OAuth to grant permission from within a browser interaction

Well, HTTP auth using your real credentials is stupid. Don't do that.

OAuth is nice, because you generate a new set of credentials during the course of granting permission, and you can revoke permission by destroying the credentials. But, OAuth can be difficult to implement. It's also overkill if all you want to do is to build a small robot or cronjob, rather than connect two web services together.

So, enter django-valet-keys. This is a way to manage credentials for robots that...

- consist of a randomized user name and password;
- can be used with HTTP Basic Auth, presumably over SSL;
- can be disabled by the human in charge at any time;
- can record and track usage.

If you *do* want to connect two web services together, this app is probably not for you. But, if you'd like something simpler for robots, this might be your thing.

CHAPTER 2

Getting Started

2.1 Installation

These instructions will be temporarily cruddy. But, here’s what I just did to get started working with this app in another site I’m developing:

First, get the package itself installed. You may find it handy to try this:

```
pip install -e 'git://github.com/lmorchard/django-valet-keys.git#egg=django-valet-keys`
```

This may or may not work, depending on whether I’ve yet done my job in building a sensible *setup.py*. (Pull requests welcome!)

2.2 Configuration

Add `valet_keys` to your `INSTALLED_APPS` list in `settings.py`:

```
INSTALLED_APPS = (
    ...
    'django.contrib.auth',
    'valet_keys',
)
```

Include `valet_keys.urls` in your site’s `urls.py`:

```
urlpatterns = patterns('',
    ...
    (r'^keys/', include('valet_keys.urls')),
    (r'^admin/', include(admin.site.urls)),
)
```

Finally, create all the models:

```
$ ./manage.py syncdb
$ ./manage.py migrate valet_keys
```

Of course, your mileage may vary, if you’re not using South to manage your model changes.

The app comes with a basic set of Django templates that you’ll undoubtedly want to copy up into your app’s template path and customize.

You’ll also probably want to link to `valet_keys.views.list` from somewhere on your site (e.g. from a user profile or settings page). That’s where valet key management starts.

CHAPTER 3

Usage

3.1 Views included

There are four views provided by django-valet-keys:

valet_keys.views.list The “home page” for django-valet-keys, listing all of a user’s keys and offering a link to create a new one. It’s handy to offer a link to this view from a profile or settings page.

valet_keys.views.new Accepts a user-authored description and generates a new key.

valet_keys.views.disable Allows a user to disable a key, linked from the list view. Note that keys are never *deleted*, only disabled. This preserves recorded history and allows later investigation of robot misbehavior.

valet_keys.views.history Paginated view of usage history for a key, linked from the list view.

Note: There’s no *edit* view. Keys can be created or disabled, but are otherwise immutable. The only user-serviceable part is the description of the key; user name and password are randomly generated.

3.2 The @accepts_valet_key decorator

In your views, the `@accepts_valet_key` decorator is the primary way to support valet keys. For example:

```
from django.http import HttpResponse
from valet_keys.decorators import accepts_valet_key

@accepts_valet_key
def hello(request):

    if request.valet_key:
        msg = 'HELLO ROBOT, I SEE YOU ARE IMPERSONATING %s' % request.user
    elif request.user.is_authenticated():
        msg = 'Why, hello there Mr. User!'
    else:
        msg = 'Welcome, guest.'

    response = HttpResponse(msg)
    response['Content-Type'] = 'text/plain'
    return response
```

The above demonstrates the features of the `@accepts_valet_key` decorator:

- `request.valet_key` is set to `None`, if the request did not present a valid key in HTTP Basic Auth.
- `request.valet_key` contains an instance of `valet_keys.models.Key` if there *was* a valid key presented via HTTP Basic Auth.
- Additionally, `request.user` is set to the Django `User` who owns the valet key.

3.3 The `Key` model and activity logging

As mentioned above, views decorated with `@accepts_valet_key` will receive an instance of `valet_keys.models.Key` in `request.valet_key` if a valid key was presented via HTTP Basic Auth.

The presence of this instance (ie. not `None`) is useful for detecting access by an authorized robot. However, there is also a logging method presented by the `Key` model object which is useful for recording what the robot did while acting on a user's behalf. It works like so:

```
@accepts_valet_key
def comment(request, slug):
    blog_post = get_object_or_404(BlogPost, slug=slug)

    if 'POST' == request.method and request.valet_key:
        content = request.POST['comment']
        blog_post.add_comment(content)
        request.valet_key.log('blog.commented', blog_post, content)
```

As demonstrated in this pseudo-code, the `Key` model offers a `log` method that accepts these parameters:

`action` Arbitrary string naming an action, handy when prefixed with an app name.

`content_object` An optional content object on which the action was performed.

`notes` Arbitrary text offering further description of the action.

Calls to `log` result in entries visible on the `valet_keys.views.history` view, thus offering rudimentary key usage tracking.

Note: This is a pretty cruddy logging API, so pull requests welcome!

Indices and tables

- *genindex*
- *search*