django-private-files Documentation

Release 0.1.1

Vasil Vangelovski (Atomidata)

February 16, 2016

Contents

This application provides utilities for controlling access to static files based on conditions you can specify within your Django application. It provides a PrivatedFileField model field and appropriate signals for monitoring access to static content. The basic goal is that you should be able to specify permissions for each PrivateFileField instance in one method (or callable) and leave the rest to django-private-files. Additionally you should be able to switch server (eg. from nginx to lighttpd) without hassle and remove this application from your project without changes to your database.

It supports the following methods for limiting access to files:

- Basic - files are served with Python (not recommended for production if you have another choice)
- Nginx - you can specify protected locations within your nginx configuration file
- xsendfile - Apache (with mod_xsendfile), lighttpd and cherokee (not tested yet)

It's currently been tested with Django 1.3, Apache, Nginx and Lighttpd. It should work with older versions of django except for the example project. Cherokee uses the same mechanism as Apache mod_xsendfile and lighttpd, so it should work, but it's not been tested or documented.

Contents:

CHAPTER 1

Installation

Install from PyPI with `easy_install` or `pip`:

```
pip install django-private-files
```

or download the source and do:

```
python setup.py install
```

or if you want to hack on the code symlink to it in your site-packages:

```
python setup.py develop
```

In your settings.py `INSTALLED_APPS` add `private_files`. You must specify a protection method (`basic`, `nginx` or `xsendfile`) in your settings.py

```
FILE_PROTECTION_METHOD = 'basic'
```

In your urls.py add the `private_files` application urls:

```
from django.conf.urls.defaults import patterns, include, url

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^private_files/', include('private_files.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
)
```

CHAPTER 2

Usage

2.1 Limiting Access to Static Files

To protect a static file that you have reference to in the database you need to use the `PrivateFileField` model field. For example:

```
from django.db import models
from private_files import PrivateFileField

class FileSubmission(models.Model):
    description = models.CharField("description", max_length = 200)
    uploaded_file = PrivateFileField("file", upload_to = 'uploads')
```

By default it will check if the user is authenticated and let them download the file as an attachment.

If you want to do more complex checks for the permission to download the file you need to pass your own callable to the `condition` parameter:

```
from django.db import models
from django.contrib.auth.models import User
from private_files import PrivateFileField

def is_owner(request, instance):
    return (not request.user.is_anonymous()) and request.user.is_authenticated and
                    instance.owner.pk = request.user.pk

class FileSubmission(models.Model):
    description = models.CharField("description", max_length = 200)
        owner = models.ForeignKey(User)
    uploaded_file = PrivateFileField("file", upload_to = 'uploads', condition = is_owner)
```

This would check if the user requesting the file is the same user referenced in the `owner` field and serve the file if it's true, otherwise it will throw `PermissionDenied`. `condition` should return `True` if the `request` user should be able to download the file and `False` otherwise.

Another optional parameter is `attachment`. It allows you to control wether the `content-disposition` header is sent or not. By default it is `True`, meaning the user will always be prompted to download the file by the browser.

2.2 Monitoring Access to Static Files

By using `django-private-files` you can monitor when a file is requested for download. By hooking to the `pre_download` signal. This fires when a user is granted access to a file and right before the server starts streaming the file to the user. The following is a simple example of using the signal to provide a download counter:

```
from django.db import models
from django.contrib.auth.models import User
from private_files import PrivateFileField, pre_download

class CountedDownloads(models.Model):
    description = models.CharField("description", max_length = 200)
    downloadable = PrivateFileField("file", upload_to = 'downloadables')
    downloads = models.PositiveIntegerField("downloads total", default = 0)

def handle_pre_download(instance, field_name, request, **kwargs):
    instance.downloads += 1
    instance.save()

pre_download.connect(handle_pre_download, sender = CountedDownloads)
```

CHAPTER 3

Server configurations

All of the bellow examples assume that:

- `MEDIA_ROOT` is set to `/media/psf/Home/Projects/django-private-files/testproject/static/`
- `MEDIA_URL` is set to `/media/`
- Protected files are stored in two subfolders `uploads` and `downloadables`
- Other static files stored in `MEDIA_ROOT` should be freely downloadable

3.1 Apache

If you serve your static content with Apache and have mod_xsendfile you can set `FILE_PROTECTION_METHOD` to `xsendfile`. Turn `XSendFile` on and deny access to the directory where you store your protected files (the value of `upload_to` appended to `MEDIA_ROOT`). Here's an exmple of a vhost configuration with mod_xsendfile and mod_wsgi:

```
<VirtualHost *:80>
        ServerName django.test
        XSendFile on
        alias /adminmedia/ /media/psf/Home/Projects/django-private-files/testproject/static/
        alias /media/ /home/vasil/src/django-trunk/django/contrib/admin/media/
        WSGIDaemonProcess django-test user=vasil group=users threads=1 processes=5
        WSGIProcessGroup django-test
        WSGIScriptAlias / /media/psf/Home/Projects/django-private-files/testproject/django.wsgi

        <Directory /media/psf/Home/Projects/django-private-files/testproject>
            Order deny,allow
            Allow from all
        </Directory>

        <Directory /media/psf/Home/Projects/django-private-files/testproject/static/uploads>
            Order deny,allow
            Deny from all
        </Directory>

        <Directory /media/psf/Home/Projects/django-private-files/testproject/static/downloadables>
            Order deny,allow
            Deny from all
        </Directory>

        <Directory /home/vasil/src/django-trunk/django/contrib/admin>
```

```
            Order deny,allow
            Allow from all
        </Directory>

    ErrorLog /var/log/httpd/test.err.log
</VirtualHost>
```

3.2 lighttpd

Lighttpd has the same mechanism of controlling access to files from a proxy backend. The following example proxies request to django running on fcgi:

```
$HTTP["host"] =~ "^django.test$" {
        server.errorlog = "/var/log/lighttpd/test-error.log"
        accesslog.filename = "/var/log/lighttpd/test-access.log"

        alias.url = (
           "/adminmedia" => "/home/vasil/src/django-trunk/django/contrib/admin/media/",
            "/media" => "/media/psf/Home/Projects/django-private-files/testproject/static/",
        )

        fastcgi.server = (
           "/django.fcgi" => (
                "main" => (
                  # Use host / port instead of socket for TCP fastcgi
                "allow-x-send-file" => "enable",
                        "host" => "127.0.0.1",
            "port" => 3033,
            "check-local" => "disable",
                )
        ),
        )

        url.access-deny = ( "/media/uploads/", "/media/downloadables/" )

        url.rewrite-once = (
                "^(/adminmedia.*)$" => "$1",
                "^(/media.*)$" => "$1",
                "^/django.fcgi(/.*)$" => "django.fcgi$1",
        "^(/.*)$" => "django.fcgi$1",
                )
}
```

3.3 Nginx

If you use nginx to serve your static files you can set the `internal` directive like so:

```
http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;

        keepalive_timeout  65;
```

```
    server {
    listen   80;
    server_name  django.test;

    location /uploads/{
        internal;
        root /media/psf/Home/Projects/django-private-files/testproject/static;
    }

    location /downloadables/{
        internal;
        root /media/psf/Home/Projects/django-private-files/testproject/static;
}


   location /media/ {
        alias /media/psf/Home/Projects/django-private-files/testproject/static/;
}


   location /media/uploads/ {
        deny all;

    }


   location /media/downloadables/ {
        deny all;
    }

    location /adminmedia {
        alias   /home/vasil/src/django-trunk/django/contrib/admin/media;
    }

    location / {
        # for a TCP host/port:
         fastcgi_pass   localhost:3033;

        # necessary parameter
        fastcgi_param PATH_INFO $fastcgi_script_name;

        include fastcgi.conf;

        # to deal with POST requests
        fastcgi_param REQUEST_METHOD $request_method;
        fastcgi_param CONTENT_TYPE $content_type;
        fastcgi_param CONTENT_LENGTH $content_length;

    }
}
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search