django-flashpolicies Documentation

Release 1.5

James Bennett

April 15, 2015

Contents

This application enables simple management of Flash cross-domain policies, which are required for Flash content to access information across domains) for Django-powered sites. Cross-domain policies are represented by an XML file format, and this application generates and serves the appropriate XML.

In the simplest case, you'll simply set up one URL pattern, pointing the URL `/crossdomain.xml` to the view `flashpolicies.views.simple()` and passing a list of domains from which you want to allow access. For example, to allow access from Flash content served from `media.example.com`, you could place the following in the root URLconf of your Django site:

```
url(r'^crossdomain.xml$',
    'flashpolicies.views.simple',
    {'domains': ['media.example.com']}),
```

CHAPTER 1

Documentation contents

1.1 Installation guide

Before installing django-flashpolicies, you'll need to have a copy of Django already installed. For information on obtaining and installing Django, consult the Django download page, which offers convenient packaged downloads and installation instructions.

The 1.5 release of django-flashpolicies officially supports Django 1.4 and 1.5; older versions may work, but are not tested or supported. Python 2.6 and 2.7 are supported for Django 1.4 and 1.5.

Additionally, on Django 1.5, django-flashpolicies 1.5 is tested and supported for Python 3.3. See *the django-flashpolicies FAQ* for additional notes on Django and Python version support.

1.1.1 Installing django-flashpolicies

There are several ways to install django-flashpolicies:

- Automatically, via a Python package installer.
- Manually, by downloading a copy of the release package and installing it yourself.
- Manually, by performing a Mercurial checkout of the latest code.

It is also highly recommended that you learn to use virtualenv for development and deployment of Python software; `virtualenv` provides isolated Python environments into which collections of software (e.g., a copy of Django, and the necessary settings and applications for deploying a site) can be installed, without conflicting with other installed software. This makes installation, testing, management and deployment far simpler than traditional site-wide installation of Python packages.

Automatic installation via a package manager

Several automatic package-installation tools are available for Python; the most popular are easy_install and pip. Either can be used to install django-flashpolicies.

Using `easy_install`, type:

```
easy_install django-flashpolicies
```

Using `pip`, type:

```
pip install django-flashpolicies
```

Manual installation from a downloaded package

If you prefer not to use an automated package installer, you can download a copy of django-flashpolicies and install it manually. The latest release package can be downloaded from django-flashpolicies' listing on the Python Package Index.

Once you've downloaded the package, unpack it (on most operating systems, simply double-click; alternately, type `tar zxvf django-flashpolicies-|version|.tar.gz` at a command line on Linux, Mac OS X or other Unix-like systems). This will create the directory `django-flashpolicies-|version|`, which contains the `setup.py` installation script. From a command line in that directory, type:

```
python setup.py install
```

Note that on some systems you may need to execute this with administrative privileges (e.g., `sudo python setup.py install`).

Manual installation from a Mercurial checkout

If you'd like to try out the latest in-development code, you can obtain it from the django-flashpolicies repository, which is hosted at Bitbucket and uses Mercurial for version control. To obtain the latest code and documentation, type:

```
hg clone http://bitbucket.org/ubernostrum/django-flashpolicies/
```

This will create a copy of the django-flashpolicies Mercurial repository on your computer; you can then the `django-flashpolicies` directory inside the checkout your Python import path, or use the `setup.py` script to perform a global installation from that code.

1.1.2 Basic configuration and use

Once installed, you can take advantage of django-flashpolicies on any Django-based site you're developing. Simply add `flashpolicies` to your `INSTALLED_APPS` setting (django-flashpolicies provides no models, so running `manage.py syncdb` is not required), and then configure one or more appropriate URL patterns to serve your cross-domain policy (or policies).

For most cases, you'll simply need a single pattern, in your root URLconf, pointing the URL `/crossdomain.xml` (the standard location for a cross-domain policy) to the view `flashpolicies.views.simple()`, passing a list of domains from which you'd like to allow access. For example, to enable access for Flash content served from the domains `media.example.com` and `api.example.com`, the following URL pattern in your root URLconf would suffice:

```
url(r'^crossdomain.xml$',
    'flashpolicies.views.simple',
    {'domains': ['media.example.com', 'api.example.com']}),
```

1.1.3 URL configuration and interaction with `APPEND_SLASH`

Your master policy file – the only policy file on your domain, in most cases – **must** be served from exactly the URL `/crossdomain.xml`. So if your site is at `example.com`, the master policy file must be served from `http://example.com/crossdomain.xml`.

As such, the Django instance in which django-flashpolicies is used must be serving from the root of the domain. If this is not possible, you will need to find an alternate method of serving your domain's cross-domain policy; one option is to manually create a `Policy` instance, and serialize it (the simplest way is via `str()`, though for more fine-grained control see the `xml_dom` attribute), writing the result to a file which can be handled normally by your web server.

If you are using Django with the CommonMiddleware enabled and the `APPEND_SLASH` setting set to `True` (by default, this is the case for any newly-created Django project), you will need to be careful in defining the URL patterns used for serving cross-domain policies. In particular, you'll want to use the regular expression `^crossdomain.xml$` – *without* trailing slash – for the URL. Django's `CommonMiddleware` will not attempt to append a slash when an existing URL pattern matches without the trailing slash.

1.2 Views for serving cross-domain policies

Included in django-flashpolicies are several views for generating and serving Flash cross-domain policies; note, however, that several of these views are for more advanced use cases and so generally are not needed. Most sites will need no more than the `simple()` policy-serving view.

`flashpolicies.views.`**`serve`**(*request*, *policy*)

Given a `flashpolicies.policies.Policy` instance, serialize it to XML and serve it. Internally, this is used by all other included views as the mechanism which actually serves the policy file.

Parameters **policy** – The `Policy` to serve.

`flashpolicies.views.`**`simple`**(*request*, *domains*)

A simple Flash cross-domain policy.

Note that if this is returned from the URL `/crossdomain.xml` on a domain, it will act as a master policy and will not permit other policies to exist on that domain. If you need to set metapolicy information and allow other policies, use the `metapolicy()` view for the master policy instead.

Parameters **domains** – A list of domains from which to allow access. Each value may be either a domain name (e.g., `example.com`) or a wildcard (e.g., `*.example.com`). Due to serious potential security issues, it is strongly recommended that you not use wildcard domain values.

`flashpolicies.views.`**`metapolicy`**(*request*, *permitted*, *domains=None*)

A Flash cross-domain policy which allows other policies to exist on the same domain.

Note that this view, if used, must be the master policy for the domain, and so must be served from the URL `/crossdomain.xml` on the domain: setting meta-policy information in other policy files is forbidden by the cross-domain policy specification.

Parameters

- **permitted** – A string indicating the extent to which other policies are permitted. *A set of constants is available, defining acceptable values for this argument*.
- **domains** – A list of domains from which to allow access. Each value may be either a domain name (e.g., `example.com`) or a wildcard (e.g., `*.example.com`). Due to serious potential security issues, it is strongly recommended that you not use wildcard domain values.

`flashpolicies.views.`**`no_access`**(*request*)

A Flash cross-domain policy which permits no access of any kind, via a metapolicy declaration disallowing all policy files.

Note that this view, if used, must be the master policy for the domain, and so must be served from the URL `/crossdomain.xml` on the domain: setting metapolicy information in other policy files is forbidden by the cross-domain policy specification.

Internally, this view simply calls the `metapolicy()` view, passing `SITE_CONTROL_NONE` as the metapolicy.

1.3 Utilities for generating cross-domain policy files

Internally, all policy files generated by django-flashpolicies are represented by instances of `flashpolicies.policies.Policy`, which understands how to handle the various permitted options in policy files and can generate the correct XML. This documentation covers `Policy` objects and their API, but is not and should not be taken to be documentation on the format and options for cross-domain policy files; Adobe's cross-domain policy specification is the canonical source for that information.

1.3.1 Simple interaction with `Policy` objects

For most cases, simply instantiating a `Policy` object with one or more domains will accomplish the desired effect. The property `xml_dom` will yield an `xml.dom.minidom.Document` object representing the policy's XML; for information on working with these objects, consult the documentation for the xml.dom.minidom module in the Python standard library. In general, however, calling `str()` with a `Policy` instance will be all that's required; this will serialize the XML to a UTF-8-encoded bytestring, suitable for writing to a file or serving over HTTP.

For example:

```
>>> from flashpolicies import policies
>>> my_policy = policies.Policy('media.example.com', 'api.example.com')
>>> print str(my_policy)
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE cross-domain-policy
  SYSTEM 'http://www.adobe.com/xml/dtds/cross-domain-policy.dtd'>
<cross-domain-policy>
        <allow-access-from domain="media.example.com"/>
        <allow-access-from domain="api.example.com"/>
</cross-domain-policy>
```

1.3.2 API reference

class `flashpolicies.policies.`**`Policy`**

Wrapper object for creating and manipulating a Flash cross-domain policy.

In the simplest case – specifying one or more domains from which to allow access – simply pass the domains to the constructor. For example:

```
my_policy = Policy('media.example.com', 'api.example.com')
```

`xml_dom`

A read-only property which returns an XML representation of this policy, as an `xml.dom.minidom.Document` object.

`allow_domain`(*domain*, *to_ports=None*, *secure=True*)

Allow access for Flash content served from a particular domain.

Parameters

- **domain** – The domain from which to allow access. May be either a full domain name (e.g., `example.com`) or a wildcard (e.g., `*.example.com`). Due to serious potential security concerns, it is strongly recommended that you avoid wildcard domain values.
- **to_ports** – (only for socket policy files) A list of ports the domain will be permitted to access. Each value in the list may be either a port number (e.g., `80`), a range of ports (e.g., `"80-120"`) or the wildcard value `"*"`, which will permit all ports.

- **secure** – If `True`, will require the security level of the HTTP protocol for Flash content to match that of this policy file; for example, if the policy file was retrieved via HTTPS, Flash content from `domain` must also be retrieved via HTTPS. If `False`, this matching of security levels will be disabled. It is strongly recommended that you not disable the matching of security levels.

allow_headers(*domain*, *headers*, *secure=True*)

Allow Flash content from a particular domain to push data via HTTP headers.

Parameters

- **domain** – The domain from which to allow access. May be either a full domain name (e.g., `example.com`) or a wildcard (e.g., `*.example.com`). Due to serious potential security concerns, it is strongly recommended that you avoid wildcard domain values.
- **headers** – A list of HTTP header names in which data may be submitted.
- **secure** – If `True`, will require the security level of the HTTP protocol for Flash content to match that of this policy file; for example, if the policy file was retrieved via HTTPS, Flash content from `domain` must also be retrieved via HTTPS. If `False`, this matching of security levels will be disabled. It is strongly recommended that you not disable the matching of security levels.

metapolicy(*permitted*)

Set metapolicy information (only applicable to master policy files), determining which other policy files may be used on the same domain.

Parameters **permitted** – The metapolicy to use. Acceptable values are those listed in the cross-domain policy specification, and are also available as *a set of constants defined in this module*. Passing an invalid value will raise `TypeError`.

By default, Flash assumes a default metapolicy of `master-only` (except for socket policies, which assume a default of `all`), so if this is the desired metapolicy (and, for security reasons, it often is), this method does not need to be called.

Note that a metapolicy of `none` forbids **all** access, even if one or more domains have previously been specified as allowed. As such, setting the metapolicy to `none` will remove all access previously granted by `allow_domain()` or `allow_headers()`. Additionally, attempting to grant access via `allow_domain()` or `allow_headers()` will, when the metapolicy is `none`, raise `TypeError`.

1.3.3 Available constants

For ease of working with metapolicies, the following constants are defined, and correspond to the acceptable values for metapolicies as defined in the cross-domain policy specification.

`flashpolicies.policies.`**SITE_CONTROL_ALL**

All policy files available on the current domain are permitted. Actual value is the string `"all"`.

`flashpolicies.policies.`**SITE_CONTROL_BY_CONTENT_TYPE**

Only policy files served from the current domain with an HTTP `Content-Type` of `text/x-cross-domain-policy` are permitted. Actual value is the string `"by-content-type"`.

`flashpolicies.policies.`**SITE_CONTROL_BY_FTP_FILENAME**

Only policy files served from the current domain as files named `crossdomain.xml` are permitted. Actual value is the string `"by-ftp-filename"`.

`flashpolicies.policies.`**SITE_CONTROL_MASTER_ONLY**

Only the master policy file for this domain – the policy served from the URL `/crossdomain.xml` – is permitted. Actual value is the string `"master-only"`.

`flashpolicies.policies.`**`SITE_CONTROL_NONE`**
No policy files are permitted, including the master policy file. Actual value is the string `"none"`.

`flashpolicies.policies.`**`VALID_SITE_CONTROL`**
A tuple containing the above constants, for convenient validation of metapolicy values.

1.4 Frequently asked questions

The following notes answer common questions, and may be useful to you when installing, configuring or using django-flashpolicies.

1.4.1 Why do I need a cross-domain policy file?

Much like JavaScript, the Adobe Flash player by default has a same-origin policy; a Flash player instance on one domain cannot load data from another domain.

A cross-domain policy file allows you, as the owner of a domain, to specify exceptions to this, allowing loading of data from another domain (for example, if you have data hosted on a CDN).

In order to prevent security issues caused by loading data from untrusted domains, your cross-domain policy file should permit *only* those domains you know are trustworthy (i.e., because those domains are under your control, and you can prevent malicious content from being placed on them).

1.4.2 What versions of Django are supported?

As of django-flashpolicies 1.5, Django 1.4 and 1.5 are supported. It is expected that django-flashpolicies 1.5 will also work unmodified with Django 1.6, once Django 1.6 is released.

Older versions of Django may work, but are not supported. In particular, the behavior of the `APPEND_SLASH` setting in old Django versions may be problematic: on some older versions of Django, `APPEND_SLASH` always adds a trailing slash even if the URL would match without it. This makes it impossible to serve a master policy file, which must have *exactly* the URL `/crossdomain.xml`, with no trailing slash.

1.4.3 What versions of Python are supported?

On Django 1.4 and 1.5, django-flashpolicies 1.5 supports Python 2.6 and 2.7. On Django 1.5, Python 3.3 is also supported.

1.4.4 Why are the elements in a different order each time I serialize my policy?

Internally, a `Policy` stores information about permitted domains and headers in dictionaries, keyed by domain names. The resulting XML is generated by iterating over these dictionaries.

In older versions of Python, iteration over a dictionary would produce the same order of keys each time provided the set of keys was identical. Newer versions of Python include a feature, for security purposes, known as hash randomization; this means that two dictionaries with the same set of keys can and will at times iterate over those keys in different orders.

Hash randomization is enabled by default on Python 3.3, and can be enabled on older releases. If you are seeing inconsistent ordering for `allow-access-from` and `allow-http-request-headers-from` elements, it is due to hash randomization being enabled.

Since this does not affect the well-formedness or validity of the resulting XML document, it is not a bug, and you should not attempt to disable hash randomization in Python.

1.4.5 Why shouldn't I use wild-card (i.e., '*') domains in my policy?

Use of wild-card entries in a policy effectively negates much of the security gain that comes from explicitly specifying the permitted domains. Unless you can and do vigilantly control all possible domains/subdomains matching a wild-card entry, use of one will expose you to the possibility of loading malicious content.

See also:

- Overview of cross-domain policy files
- Policy file format specification
- Adobe's recommendations for use of Flash cross-domain policies

f

A

F

M

N

P

S

V

X