django-browserid Documentation

Release 0.6

Paul Osman, Michael Kelly

October 10, 2012

CONTENTS

django-browserid is a library that integrates BrowserID authentication into Django.

django-browserid provides an authentication backend, `BrowserIDBackend`, that verifies BrowserID assertions using the browserid.org verification service and authenticates users. It also provides `verify`, which lets you build more complex authentication systems based on BrowserID.

django-browserid is a work in progress. Contributions are welcome. Feel free to fork and contribute!

CHAPTER

ONE

SETUP

1.1 Installation

You can use pip to install django-browserid and requirements:

```
pip install django-browserid
```

1.2 Configuration

To use `django-browserid`, add it to `INSTALLED_APPS` in `settings.py`:

```
INSTALLED_APPS = (
    # ...
    'django.contrib.auth',
    'django_browserid',  # Load after auth to monkey-patch it.
    # ...
)
```

and add `django_browserid.auth.BrowserIDBackend` to `AUTHENTICATION_BACKENDS` in `settings.py`:

```
AUTHENTICATION_BACKENDS = (
    # ...
    'django_browserid.auth.BrowserIDBackend',
    # ...
)
```

Edit your `urls.py` file and add the following:

```
urlpatterns = patterns('',
    # ...
    (r'^browserid/', include('django_browserid.urls')),
    # ...
)
```

You should also add the following in `settings.py`:

```
# Note: No trailing slash
SITE_URL = 'https://example.com:8000'
```

BrowserID uses an assertion and an audience to verify the user. This `SITE_URL` is used to determine the audience. For security reasons, it is *very important* that you set `SITE_URL` correctly.

You can also set the following optional config in `settings.py` (they have sensible defaults):

```
# Path to redirect to on successful login.
LOGIN_REDIRECT_URL = '/'

# Path to redirect to on unsuccessful login attempt.
LOGIN_REDIRECT_URL_FAILURE = '/'
```

Somewhere in one of your templates, you'll need to create a link and a form with a single hidden input element, which you'll use to submit the BrowserID assertion to the server. If you want to use `django_browserid.forms.BrowserIDForm`, you could use something like the following template snippet:

```
{% if not user.is_authenticated %}
<a id="browserid" href="#">Sign In</a>
<form method="POST" action="{% url browserid_verify %}">
   {% csrf_token %}
   {{ browserid_form.as_p }}
</form>
{% endif %}
```

If you use browserid_form, it is further recommended that you add `django_browserid.context_processors.browserid_f` to `TEMPLATE_CONTEXT_PROCESSORS`; this will create the `browserid_form` variable automatically in `RequestContext` instances when needed. That is, in `settings.py`:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    # ...
    'django_browserid.context_processors.browserid_form',
    # ...
)
```

You will also need to include JavaScript to power the BrowserID popup and form. You can use django form media at the bottom of your page (see Form Media and Managing static files for more information):

```
{{ browserid_form.media }}
```

This JavaScript file requires jQuery.

Note: If you don't want to use the static files framework, you'll need to include the `https://browserid.org/include.js` file, as well as JavaScript similar to `django_browserid/static/browserid/browserid.js`:

```
<script src="https://browserid.org/include.js"></script>
<!-- Include JS for browserid_form here. -->
```

Note: If your site uses Content Security Policy, you will have to add directives to allow the external browserid.org JavaScript, as well as an iframe used as part of the login process.

If you're using django-csp, the following settings will work:

```
CSP_SCRIPT_SRC = ("'self'", 'https://browserid.org',)
CSP_FRAME_SRC = ("'self'", 'https://browserid.org',)
```

CHAPTER

TWO

ADVANCED USAGE

2.1 Automatic Account Creation

`django-browserid` will automatically create a user account for new users if the setting `BROWSERID_CREATE_USER` is set to `True` in `settings.py`. The user account will be created with the verified email returned from the BrowserID verification service, and a URL safe base64 encoded SHA1 of the email with the padding removed as the username.

To provide a customized username, you can provide a different algorithm via your settings.py:

```
# settings.py
BROWSERID_CREATE_USER = True
def username(email):
    return email.rsplit('@', 1)[0]
BROWSERID_USERNAME_ALGO = username
```

You can can provide your own function to create users by setting `BROWSERID_CREATE_USER` to a string path pointing to a function:

```
# module/util.py
def create_user(email):
    return User.objects.create_user(email, email)

# settings.py
BROWSERID_CREATE_USER = 'module.util.create_user'
```

You can disable account creation, but continue to use the `browserid_verify` view to authenticate existing users with the following:

```
BROWSERID_CREATE_USER = False
```

2.2 Custom Verification

If you want full control over account verification, don't use django-browserid's `browserid_verify` view. Create your own view and use `verify` to manually verify a BrowserID assertion with something like the following:

```
from django_browserid import get_audience, verify
from django_browserid.forms import BrowserIDForm


def myview(request):
    # ...
```

```
    if request.method == 'POST':
        form = BrowserIDForm(data=request.POST)
        if not form.is_valid():
            result = verify(form.cleaned_data['assertion'], get_audience(request))
            if result:
                # check for user account, create account for new users, etc
                user = my_get_or_create_user(result.email)
```

`result` will be `False` if the assertion failed, or a dictionary similar to the following:

```
{
    u'audience': u'https://mysite.com:443',
    u'email': u'myemail@example.com',
    u'issuer': u'browserid.org',
    u'status': u'okay',
    u'expires': 1311377222765
}
```

You are of course then free to store the email in the session and prompt the user to sign up using a chosen identifier as their username, or whatever else makes sense for your site.

2.3 Javascript Fallback

It is a good idea to provide an alternative method of authenticating with your site for users that do not have JavaScript available. An easy way of doing this is to modify the `href` of the link that you bind to BrowserID login to point to a traditional login and registration page:

```
<a id="browserid" href="{% url 'login.view.name' %}">Sign In</a>
```

If a user has JavaScript enabled, when they click the link the JavaScript will take over and show a BrowserID popup. If a user has JavaScript disabled, they will be directed to your login view (which should not require JavaScript, of course).

2.4 Signals

`django_browserid.signals.`**`user_created`**

Signal triggered when a user is automatically created during authentication.

- **sender**: The function that created the user instance.
- **user**: The user instance that was created.

CHAPTER

THREE

SETTINGS

`django.conf.settings.`**`LOGIN_REDIRECT_URL`**
Default: `'/accounts/profile'`

Path to redirect to on successful login. If you don't specify this, the default Django value will be used.

`django.conf.settings.`**`LOGIN_REDIRECT_URL_FAILURE`**
Default: `'/'`

Path to redirect to on an unsuccessful login attempt.

`django.conf.settings.`**`BROWSERID_CREATE_USER`**
Default: `True`

If `True` or `False`, enables or disables automatic user creation during authentication.

If set to a string, it is treated as an import path pointing to a custom user creation function. See *auto-user* for more information.

`django.conf.settings.`**`BROWSERID_VERIFICATION_URL`**
Default: `'https://browserid.org/verify`

Defines the URL for the BrowserID verification service to use.

`django.conf.settings.`**`BROWSERID_DISABLE_CERT_CHECK`**
Default: `False`

Disables SSL certificate verification during BrowserID verification. *Never disable this in production!*

`django.conf.settings.`**`BROWSERID_CACERT_FILE`**
Default: `None`

CA cert file used during validation. If none is provided, the default file included with requests is used.

CHAPTER

FOUR

TROUBLESHOOTING

4.1 CSP WARN: Directive "..." violated by https://browserid.org/include.js

This warning appears in the Error Console when your site uses Content Security Policy without making an exception for the browserid.org external JavaScript include.

To fix this, include https://browserid.org in your script-src directive. If you're using the django-csp library, the following settings will work:

```
CSP_SCRIPT_SRC = ("'self'", 'https://browserid.org',)
CSP_FRAME_SRC = ("'self'", 'https://browserid.org',)
```

Note: The example above also includes the frame-src directive. There is an iframe used during BrowserID login, but some people report that login will work without the directive. In general, you should probably include it.

4.2 Login fails silently due to SESSION_COOKIE_SECURE

If you try to login on a local instance of a site and login fails without any error (typically redirecting you back to the login page), check to see if you've set *SESSION_COOKIE_SECURE* to True in your settings.

SESSION_COOKIE_SECURE controls if the *secure* flag is set on the session cookie. If set to True on a local instance of a site that does not use HTTPS, the session cookie won't be sent by your browser because you're using an HTTP connection.

The solution is to set *SESSION_COOKIE_SECURE* to False on your local instance, typically by adding it to *settings/local.py*:

```
SESSION_COOKIE_SECURE = False
```

4.3 Login fails silently due to cache issues

Another possible cause of silently failing logins is an issue with having no cache configured locally. Several projects (especially projects based on playdoh, which uses django-session-csrf) store session info in the cache rather than the database, and if your local instance has no cache configured, the session information will not be stored and login will fail silently.

To solve this issue, you should configure your local instance to use an in-memory cache with the following in your local settings file:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake'
    }
}
```

PYTHON MODULE INDEX

d